System Verification and Runtime Monitoring with Multiple Weakly-Hard Constraints

YI-TING HSIEH, TZU-TAO CHANG, CHEN-JUN TSAI, SHIH-LUN WU, CHING-YUAN BAI, KAI-CHIEH CHANG, and CHUNG-WEI LIN, National Taiwan University, Taiwan
EUNSUK KANG, Carnegie Mellon University, USA
CHAO HUANG, University of Liverpool, UK
QI ZHU, Northwestern University, USA

A weakly-hard fault model can be captured by an (m, k) constraint, where $0 \leq m \leq k$, meaning that there are at most m bad events (faults) among any k consecutive events. In this paper, we use a weakly-hard fault model to constrain the occurrences of faults in system inputs. We develop approaches to verify properties for all possible values of (m, k), where k is smaller than or equal to a given K, in an exact and efficient manner. By verifying all possible values of (m, k), we define weakly-hard requirements for the system environment and design a runtime monitor based on counting the number of faults in system inputs. If the system environment satisfies the weakly-hard requirements, the satisfaction of desired properties is guaranteed; otherwise, the runtime monitor can notify the system to switch to a safe mode. This is especially essential for cyber-physical systems which need to provide guarantees with limited resources and the existence of faults. Experimental results with discrete second-order control, network routing, vehicle following, and lane changing demonstrate the generality and the efficiency of the proposed approaches.

CCS Concepts: • **Computer systems organization → Embedded and cyber-physical systems**; **Real-time systems**; **Dependable and fault-tolerant systems and networks**; • **Security and privacy** → *Systems security*.

Additional Key Words and Phrases: Formal verification, runtime monitoring, weakly-hard models

ACM Reference Format:
Yi-Ting Hsieh, Tzu-Tao Chang, Chen-Jun Tsai, Shih-Lun Wu, Ching-Yuan Bai, Kai-Chieh Chang, Chung-Wei Lin, Eunsuk Kang, Chao Huang, and Qi Zhu. 2023. System Verification and Runtime Monitoring with Multiple Weakly-Hard Constraints. 1, 1 (July 2023), 28 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

A weakly-hard fault model can be captured by an (m, k) constraint, where $0 \leq m \leq k$, meaning that there are at most m bad events (faults) among any k consecutive events. Weakly-hard models have been studied in a number of works for real-time systems [1–3, 5, 6, 10, 11, 13, 19, 20, 27, 28], mostly from the perspective of scheduling constraints. In this paper, we use a weakly-hard model to

Authors' addresses: Yi-Ting Hsieh, r09922082@ntu.edu.tw; Tzu-Tao Chang, b05703092@ntu.edu.tw; Chen-Jun Tsai, r11922055@ntu.edu.tw; Shih-Lun Wu, b06902080@ntu.edu.tw; Ching-Yuan Bai, b05502055@ntu.edu.tw; Kai-Chieh Chang, r08922054@ntu.edu.tw; Chung-Wei Lin, cwlin@csie.ntu.edu.tw, National Taiwan University, No. 1, Sec. 4, Roosevelt Rd., Taipei, , Taiwan, 10617; Eunsuk Kang, Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, Pennsylvania, USA,, eunsukk@andrew.cmu.edu; Chao Huang, University of Liverpool, Liverpool, UK, L69 3BX, chao.huang2@liverpool.ac.uk; Qi Zhu, Northwestern University, 2145 Sheridan Road, Evanston, IL, USA,, qzhu@northwestern.edu.

XXXX-XXXX/2023/7-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

constrain the occurrences of faults, and verify properties of discrete systems under such weakly-hard fault model.

1.1 Motivating Applications and Usages

Verifying system properties under the weakly-hard fault model has various applications, such as:

- In a real-time system, a deadline miss can be considered as a bad event (fault). Our approaches can help find the maximum number of deadline misses allowed while guaranteeing system properties, which can then be used to reduce computation/communication load and maximize resource saving (*e.g.*, CPU or network resource) with a less critical mode of the system.
- In a networked system, a message without authentication can be modeled as a bad event (fault), and again, our approaches can be applied to maximize resource saving (*e.g.*, reduce the computation and transmission of message authentication codes) by allowing messages without authentication, while still ensuring system properties. Note that a system that only authenticates partial messages has also been proposed [17].
- In the systems above, a deadline miss (*e.g.*, due to a denial-of-service attack) or a compromised message can be caused by a malicious attacker. From the perspective of an attacker, our approaches can be applied to minimize attack cost while still causing the system to reach a state violating properties.

More generally speaking, our verification approaches under the (m, k) weakly-hard fault model bring important usages for system engineering of cyber-physical systems which need to provide guarantees with limited resources and the existence of faults:

- If the environment and system design (*e.g.*, via scheduling) ensure that the fault occurrences satisfy the (m, k) constraint, the system properties are satisfied.
- If the environment and system design cannot ensure that the fault occurrences always satisfy the (m, k) constraint, a runtime monitor should be developed to monitor the occurrences of faults and adapt the system to a safe (more conservative) mode when the (m, k) constraint is violated.

For example, applications of connected vehicles, such as intersection management and cooperative adaptive cruise control, rely on periodic messages from other vehicles or roadside units. However, a message may be missing due to network faults or even malicious attacks. With the verification results, a connected vehicle can monitor the number of missing messages during runtime. If the corresponding (m, k) constraint is violated, the connected vehicle should switch to a safe mode (*e.g.*, slowing down or stopping immediately). It should be emphasized that, in practice, the cost of a network without missing messages is too high, or even it may not be possible to predict how the environment behaves, so the satisfaction of the (m, k) constraint cannot be guaranteed. Therefore, a runtime monitor for the (m, k) constraint is desired.

1.2 Target Problem and Contributions

In this paper, given a labelled transition system S, a property P, and a positive integer K, we aim to find a subset of the (m, k) constraints, where $1 \leq m \leq k \leq K$, such that, if the environment satisfies at least one constraint in the subset, then S guarantees to satisfy P; otherwise, if the environment does not satisfy any constraint in the subset, then S cannot guarantee to satisfy P, which should lead S to switch to a safe mode. Based on the subset, we can then develop a runtime monitor to verify whether the environment satisfies at least one constraint in the subset. Unlike some existing runtime-monitoring approaches (without an explicit model of S), this paper assumes that the model of S is given, but the satisfaction of an (m, k) constraint can only be verified during runtime.

Table 1. The overview of the proposed approaches.

Property & System	Single (m,k) Constraint	Multiple (m,k) Constraints
Reachability & Finite-State Machine	Mask-Compressing (Section 5.2)	Layered BFS, Dual-Layered BFS (Sections 5.3 and 5.4)
General Property & General System	Not Covered	Algorithms 1, 2, and 3 (Sections 3.2, 3.3, and 3.4)

The runtime monitor relies on a *safety table* which stores the satisfaction condition of property P under each (m,k) constraint. Since $1 \leq m \leq k \leq K$, there are $\frac{K(K+1)}{2}$ possible (m,k) pairs and thus $\frac{K(K+1)}{2}$ constraints in the safety table. A straightforward approach evaluating each (m,k) constraint one by one needs to verify the property $\frac{K(K+1)}{2}$ times, where each individual verification may be expensive to carry out. To remedy this problem, we propose approaches to compute the safety table in a more efficient way.

The main contributions of this paper include[1]:

- We derive theorems that state various logical relationships between weakly-hard constraints. Based on these relationships, we reduce the problem of computing a safety table to that of computing its *satisfaction boundary* and propose approaches that require verifying at most $2K$ times to compute this satisfaction boundary.
- Based on the resulting satisfaction boundary, we define weakly-hard requirements for the system environment and design a lightweight runtime monitor that dynamically checks the satisfaction of the weakly-hard requirements.
- We prove that, without being given a satisfaction boundary as an input, an optimal deterministic approach does not exist. Then, given a satisfaction boundary as an input, we introduce an optimal approach which can be used to appraise the efficiency of the proposed approaches. The correctness of the optimal approach and the uniqueness of its evaluated weakly-hard constraints.
- We consider a special case of reachability of finite-state machines. We propose a layered Breadth-First Search (BFS) approach which computes the satisfaction boundary for all (m,k) constraints ($1 \leq m \leq k \leq K$) with the same computational complexity as evaluating a single (m,K) constraint (K is the upper bound of all possible values of k). We further propose a dual-layered BFS approach to make the computation more efficient.
- Experimental results with discrete second-order control, network routing, and lane changing demonstrate the generality and the efficiency of the proposed approaches.

1.3 Overview of Proposed Approaches and Paper Organization

We overview the proposed approaches in this paper in Table 1. There are six approaches: the monotonic approach (Algorithm 1) in Section 3.2, the monotonic approach with dynamic upper bound of satisfaction boundary (Algorithm 2) in Section 3.3, the lowest-cast-first heuristic (Algorithm 3) in Section 3.4, the mask-compressing approach in Section 5.2, the layered BFS approach in Section 5.3,

[1]The preliminary version of this paper is published at the 2020 International Conference on Runtime Verification (RV) [26]. This journal version adds new technical contents as follows. (1) We consider the existence of an optimal deterministic approach. (2) We refine the implementation of the layered Breadth-First Search (BFS) approach to make it more efficient. (3) We propose a dual-layered BFS approach which is more efficient than the layered BFS approach. (4) We consider two more applications, network routing and lane changing, to demonstrate the applicability and scalability of weakly-hard fault modeling and the proposed approaches.

and the dual-layered BFS approach in Section 5.4. The first three approaches are for general properties, general systems, and multiple weakly-hard constraints, with different evaluation orders. They decide the orders of evaluating the weakly-hard constraints and need to call a verification approach for a single weakly-hard constraint. Note that the first three approaches assume that one can verify a property P under a single weakly-hard constraint — this paper does not cover how to achieve that, except in the special case of reachability for finite-state machines. The last three approaches are exactly for the special case of reachability for finite-state machines. The mask-compressing approach is for a single weakly-hard constraint, and thus it can be plugged into (called by) the first three approaches, while the layered BFS approach and the dual-layered BFS approach are for multiple weakly-hard constraints.

The paper is organized as follows. Section 2 provides the problem formulation. Section 3 describes how we solve the problem for general properties and systems and introduces a runtime monitor. Section 4 discusses optimal approaches. Section 5 considers the special case of reachability for finite-state machines. Section 6 presents the experimental results. Section 7 reviews the related work, and Section 8 concludes the paper.

2 PROBLEM FORMULATION

In this paper, we consider a labelled transition system $S = \langle Q, \Sigma, R, Q_0 \rangle$ where Q is the set of states, Σ is the set of alphabet, $R \subseteq Q \times \Sigma \times Q$ is the transition relation, and $Q_0 \subseteq Q$ is the set of initial states. Without loss of generality, a subset of the alphabet represents input events $\{0, 1\} \subseteq \Sigma$, where 0 and 1 represent a normal and faulty environmental event, respectively. We use $\sigma \in \Sigma^* = \{0, 1\}^*$ to represent an input trace. We are interested in evaluating whether a property P is satisfied with inputs under the constraints of weakly-hard fault models. As mentioned in the overview in Section 1.3, we discuss a general property P, which can be defined by any property specification language, in Section 3 and the reachability property in Section 5.

Definition 1. ***Weakly-Hard Fault Model***. *A weakly-hard fault model is defined by (m, k), meaning that there are at most m faulty events (denoted as 1's) among any k consecutive events in the input trace. The corresponding constraint is denoted as $W(m, k)$.*

Based on the definition, an input trace $\sigma \models W(m, k)$ if and only if σ has at most m 1's in any size-k window of σ.

Definition 2. ***Weakly-Hard Constraint Set***. *Given $K \in \mathbb{Z}^+$, the weakly-hard constraint set is defined as $C(K) := \{W(m, k) \mid 1 \leq m \leq k \leq K\}$.*

Given a system S, a property P, and a positive integer K, the goal of the approaches in the following sections is to find a subset of $C(K)$, such that, if the environment satisfies at least one constraint in the subset, then S guarantees to satisfy P; otherwise, if the environment does not satisfy any constraint in the subset, then S cannot guarantee to satisfy P, which should lead S to switch to a safe mode. We do not consider the case of $m = 0$ as, if there is no faulty event, S should be designed to satisfy P, which should be regarded as a design-time problem (although our approach can also be used to handle this special case).

The subset that leads S to guarantee P is stored in a *safety table* which keeps the satisfaction condition of P under each $W(m, k)$ in $C(K)$. A safety table is defined as follows.

Definition 3. ***Safety Table***. *Given $K \in \mathbb{Z}^+$, a safety table $T \in \{True, False, N/A\}^{K \times K}$ is defined as*

$$T[m, k] = \begin{cases} True & \text{if } m \leq k \text{ and } \forall \sigma \models W(m, k),\ S \models P; \\ False & \text{if } m \leq k \text{ and } \exists \sigma \models W(m, k),\ S \not\models P; \\ N/A & \text{if } m > k. \end{cases} \tag{1}$$

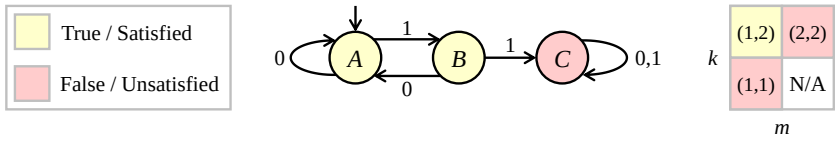

Fig. 1. (a) An example safety table and (b) its satisfaction boundary.

For $m > k$, $T[m,k]$ is not applicable as the corresponding weakly-hard fault model is undefined. All $W(m,k)$, where $T[m,k]$ = True, exactly forms the subset that leads S to guarantee P. $T[m,k]$ is determined when the property P under $W(m,k)$ is verified or falsified. Computing the safety table is the goal of the approaches in the following sections, and the safety table is the results (outputs) of them.

Based on the safety table, we can then develop a runtime monitor to verify whether the environment satisfies at least one constraint in the subset that leads S to guarantee P. Note that a safety table is computed off-line in the design phase, and the satisfaction of P under each $W(m,k)$ in $C(K)$ needs to be stored and accessed during runtime.

An example is shown in Figure 1. Given a system S as a finite-state machine in Figure 1(a), state A is the initial state, states A and B are safe, state C is unsafe, and the property P is that S must always be in a safe state. The corresponding safety table is in Figure 1(b). $T[1,1]$ and $T[2,2]$ are False because it is possible that an input trace has all 1's and leads S to reach the unsafe state C. On the other hand, $T[1,2]$ is True because any input trace has at most one 1's in any size-2 window so that S will only reach the safe states A and B. A larger example safety table is shown in Figure 2(a).

3 GENERAL APPROACHES AND RUNTIME MONITOR DESIGN

In this section, we first define the strength of weakly-hard constraints (Section 3.1). We then derive the fundamental theorems of logical relationships between weakly-hard constraints and propose an algorithm to compute the safety table and its corresponding satisfaction boundary based on these theorems (Section 3.2). We further derive advanced theorems of logical relationships between weakly-hard constraints and propose an improved algorithm (Section 3.3) and a lowest-cost-first heuristic (Section 3.4) taking all properties into account. Based on the computed safety table and the satisfaction boundary, we can design a runtime monitor (Section 3.5).

3.1 Strength of Weakly-Hard Constraint

Definition 4. ***Strength of Weakly-Hard Constraints***. *Given two weakly-hard constraints $W(m,k)$ and $W(m',k')$, we define that $W(m,k)$ is stronger than $W(m',k')$, denoted as $W(m,k) \succ W(m',k')$, if and only if any input trace that satisfies $W(m,k)$ also satisfies $W(m',k')$.*

Understanding the logical relationships between constraints allows us to determine the satisfaction of properties under some $W(m,k)$ constraints directly from the known verification results of other $W(m',k')$ constraints. From an algorithm design perspective, exploiting these relationships by evaluating the constraints in a proper order leads to a significant improvement in efficiency.

3.2 Monotonic Approach

Theorem 1. *For any $m, m', k \in \mathbb{Z}^+, m < m' \leq k$, $W(m,k) \succ W(m',k)$.*

Proof. By definition, for any input trace $\sigma \models W(m,k)$, it has at most m 1's in any size-k window of σ. Since $m < m'$, it follows that $\sigma \models W(m',k)$. □

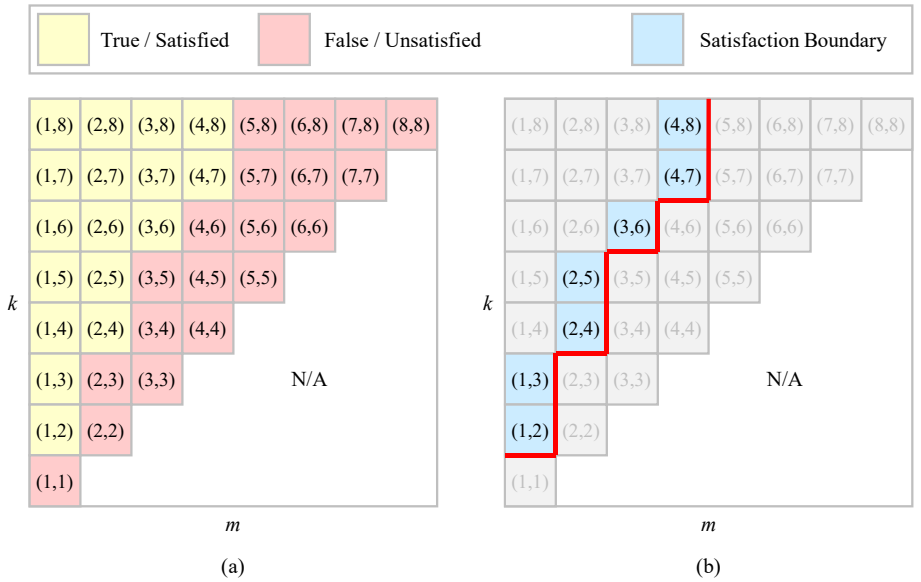

Fig. 2. (a) An example safety table and (b) its satisfaction boundary.

Implication 1. *For any $m, m', k \in \mathbb{Z}^+, m < m' \leq k$, if a property P is unsatisfied under $W(m,k)$, then P is unsatisfied under $W(m',k)$; if a property P is satisfied under $W(m',k)$, then P is satisfied under $W(m,k)$.*

Theorem 2. *For any $m, k, k' \in \mathbb{Z}^+, m \leq k' < k$, $W(m,k) \succ W(m,k')$.*

Proof. By definition, for any input trace $\sigma \models W(m,k)$, it has at most m 1's in any size-k window of σ. If we reduce the window size to k', the maximum number of 1's in the window only remains the same or decreases, so it follows that $\sigma \models W(m,k')$. □

Implication 2. *For any $m, k, k' \in \mathbb{Z}^+, m \leq k' < k$, if a property P is unsatisfied under $W(m,k)$, then P is unsatisfied under $W(m,k')$; if a property P is satisfied under $W(m,k')$, then P is satisfied under $W(m,k)$.*

By Implication 1, the problem of computing a safety table can be reduced to the problem of computing the *satisfaction boundary* of the safety table. The satisfaction boundary is defined as follows.

Definition 5. ***Satisfaction Boundary***. *For each k, the satisfaction boundary $B(k)$ is the maximum m such that $T[m,k]$ (in the safety table) is True.*

The satisfaction boundary of the safety table in Figure 2(a) is shown in Figure 2(b). The reduction is crucial because we only need to store the satisfaction boundary rather than the whole safety table for the runtime monitor.

Implications 1 and 2 imply that evaluating constraints in a monotonic manner (*i.e.*, increasing m and increasing k until a given K) can compute the satisfaction boundary without evaluating all constraints in $C(K)$. We assume that we can verify a property P under a single $W(m,k)$ — an example of verifying reachability under a single $W(m,k)$ is described in Section 5. We propose the monotonic approach (Algorithm 1) to compute the satisfaction boundary $B(k)$ for each $k \leq K$. For each $k \leq K$, the algorithm increases m until P is unsatisfied and obtains $B(k)$ (Lines 5–11). By

Algorithm 1 Monotonic Approach

```
1:  procedure GET_SATISFACTION_BOUNDARY(S, P, K)
2:      B ← [ ]
3:      m ← 0
4:      for k ← 1 to K do                          ▷ Get satisfaction boundary for each k
5:          while m < k do
6:              if S ⊭ P under W(m + 1, k) then
7:                  break
8:              end if
9:              m ← m + 1
10:         end while
11:         B[k] ← m
12:     end for
13:     return B
14: end procedure
```

Implication 1, since P is unsatisfied under $W(B(k)+1,k)$, P is unsatisfied under $W(m,k)$ where $m > B(k)+1$, and thus there is no need to verify P under $W(m,k)$ where $m > B(k)+1$. For example, as shown in Figure 3(a), if P is unsatisfied under $W(3,4)$, then P is unsatisfied under $W(4,4)$, which does not need to be evaluated. Then, k is increased by 1 (Line 4), and the same procedure repeats and starts with $m = B(k-1)+1$ (not $m = 1$). By Implication 2, since P is satisfied under $W(B(k-1),k-1)$, P is satisfied under $W(B(k-1),k)$, and thus there is no need to verify P under $W(B(k-1),k)$. For example, as shown in Figure 3(b), if P is satisfied under $W(3,4)$, then P is satisfied under $W(3,5)$ (and $W(3,k)$ where $k \geq 5$), which does not need to be evaluated. The algorithm terminates when $B(k)$ is computed for each $k \leq K$, and the satisfaction boundary is returned (Line 13).

Assuming the complexity of verifying P under a single weakly-hard constraint is $O(X)$, the complexity of Algorithm 1 is $O(2K \cdot X) = O(K \cdot X)$, since both m, k are non-decreasing in the algorithm and bounded above by K. It is a significant improvement over brute-forcing each $W(m,k)$ in $C(K)$, which has the complexity $O(K^2 \cdot X)$.

3.3 Monotonic Approach with Dynamic Upper Bound of Satisfaction Boundary

THEOREM 3. *For any $m, k, x \in \mathbb{Z}^+, m < k, x \geq 2$, $W(m,k) > W(xm, xk)$.*

PROOF. For any input trace $\sigma \models W(m,k)$ and size-(xk) window of σ, the window can be constructed by x size-k windows, and each of which has at most m 1's. Thus, there are at most xm 1's in the size-(xk) window, and it follows that $\sigma \models W(xm, xk)$. □

IMPLICATION 3. *For any $m, k, x \in \mathbb{Z}^+, m < k, x \geq 2$, if a property P is unsatisfied under $W(m,k)$, then P is unsatisfied under $W(xm, xk)$; if a property P is satisfied under $W(xm, xk)$, then P is satisfied under $W(m,k)$.*

THEOREM 4. *For any $m, k, x \in \mathbb{Z}^+, m < k$, $W(m,k) > W(m+x, k+x)$.*

PROOF. For any input trace $\sigma \models W(m,k)$ and size-$(k+x)$ window of σ, the window can be constructed by combining two windows of sizes k and x, respectively. Since $\sigma \models W(m,k)$, there are at most m 1's in the size-k window. On the other hand, there are at most x 1's in the size-x window. Thus, there are at most $(m+x)$ 1's in the size-$(k+x)$ window, and it follows that $\sigma \models W(m+x, k+x)$. □

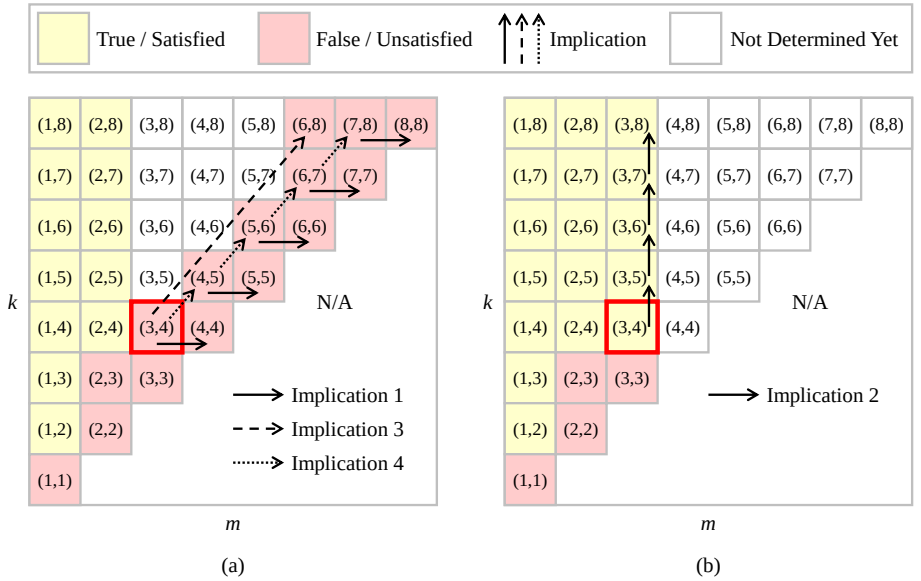

Fig. 3. An illustration of Algorithms 1 (which applies Implications 1 and 2 only) and 2 (which applies Implications 1, 2, 3, and 4). To have a clear comparison, we focus on the implications of $W(3,4)$ only. (a) If P is unsatisfied under $W(3,4)$, then P is unsatisfied under $W(4,4)$. Algorithm 2 further implies that P is unsatisfied under $W(6,8)$ and $W(m,k)$ where $k \geq 5$ and $m \geq k-1$. (b) If P is satisfied under $W(3,4)$, then P is satisfied under $W(3,k)$ where $k \geq 5$.

Implication 4. *For any $m, k, x \in \mathbb{Z}^+, m < k$, if a property P is unsatisfied under $W(m,k)$, then P is unsatisfied under $W(m+x, k+x)$; if a property P is satisfied under $W(m+x, k+x)$, then P is satisfied under $W(m,k)$.*

Implications 3 and 4 imply the satisfaction of a property P beyond the same m or k. Integrating with the previously proposed monotonic approach which increases m and k, we exploit the implications and propose the monotonic approach with dynamic upper bound of satisfaction boundary (Algorithm 2) to compute the satisfaction boundary $B(k)$ for each $k \leq K$. The main difference between Algorithms 1 and 2 is that the former one considers the search range for the satisfaction boundary from an m to k, while the latter one dynamically reduces the search range whenever P is unsatisfied under a constraint.

Specifically, suppose the algorithm is in the process of computing $B(k)$, and P is unsatisfied under $W(m+1,k)$ (Line 9). By Implication 3, P is unsatisfied for each $W(x \cdot (m+1), xk), x \geq 2$, and thus $x \cdot (m+1) - 1$ is an upper bound of $B(xk)$ (Lines 10–14). Similarly, by Implication 4, P is unsatisfied for each $W((m+1)+x, k+x), x \in \mathbb{Z}^+$, and thus $(m+1)+x-1$ is an upper bound of $B(k+x)$ (Lines 15–19). An example is shown in Figure 3(a), if P is unsatisfied under $W(3,4)$, then P is unsatisfied under $W(4,4)$, $W(6,8)$, and $W(m,k)$ where $k \geq 5$ and $m \geq k-1$, and they do not need to be evaluated. If P is satisfied under $W(3,4)$, then the implication is the same as Algorithm 1, as shown in Figure 3(b).

3.4 Lowest-Cost-First Heuristic

Since the implications of the theorems do not necessarily restrict the order of evaluating each $W(m,k)$ in $C(K)$, the efficiency can be further improved by a good evaluation order. We suppose

Algorithm 2 Monotonic Approach with Dynamic Upper Bound of Satisfaction Boundary

```
1: procedure GET_SATISFACTION_BOUNDARY(S, P, K)
2:     B ← [ ]
3:     m ← 0
4:     for k ← 1 to K do                          ▷ Initialize satisfaction boundary
5:         B[k] = k
6:     end for
7:     for k ← 1 to K do                          ▷ Get satisfaction boundary for each k
8:         while m < B[k] do
9:             if S ⊭ P under W(m + 1, k) then
10:                x ← 2
11:                while x · k ≤ K do                          ▷ Implication 3
12:                    B[xk] ← min(B[xk], x · (m + 1) − 1)
13:                    x ← x + 1
14:                end while
15:                x ← 1
16:                while k + x ≤ K do                          ▷ Implication 4
17:                    B[k + x] ← min(B[k + x], (m + 1) + x − 1)
18:                    x ← x + 1
19:                end while
20:                break
21:            end if
22:            m ← m + 1
23:        end while
24:        B[k] ← min(B[k], m)
25:    end for
26:    return B
27: end procedure
```

Algorithm 3 Lowest-Cost-First Heuristic

```
1: procedure GET_SAFETY_TABLE(S, P, K)
2:     T ← {undefined}                    ▷ Initialize as undefined for the safety table
3:     while T has undefined element do
4:         Select the lowest-cost undefined W(m, k)
5:         if S ⊨ P under W(m, k) then
6:             T[m, k] ← True
7:         else
8:             T[m, k] ← False
9:         end if
10:        Recursively update T by Implications 1, 2, 3, and 4
11:    end while
12:    return T
13: end procedure
```

that we can estimate the verification (time) cost for each $W(m,k)$ in $C(K)$, *e.g.*, based on the complexity as a function of m and k. Intuitively, evaluating lower-cost constraints which implies

Algorithm 4 Runtime Monitoring

```
1:  procedure RUNTIME_MONITORING(K, B[])
2:      for k ← 1 to K do
3:          I[k] ← 0                                  ▷ Store the last k-th input
4:          N_1[k] ← 0                ▷ Store the number of 1's among the last k inputs
5:      end for
6:      i ← 0
7:      while 1 do                                                ▷ During runtime
8:          x = Get_Input()
9:          for k ← 1 to K do
10:             N_1[k] ← N_1[k] + x − I[(i − k)%K]
11:             if N_1[k] > B[k] then                ▷ Exceed the satisfaction boundary
12:                 Switch to a safe mode
13:             end if
14:         end for
15:         I[i] ← x
16:         i ← (i + 1)%K
17:     end while
18: end procedure
```

more constraints or higher-cost constraints is preferred. We propose the lowest-cost-first heuristic (Algorithm 3) which iteratively selects a not-yet-evaluated constraint in $C(K)$ by the estimated cost (Line 4), evaluates it (Lines 5–9), and processes all implied constraints after each evaluation (Line 10). The lowest-cost-first heuristic, though not optimal, provides the flexibility of evaluating constraints in $C(K)$ by different orders. The lowest-cost-first heuristic, though not optimal, provides the flexibility of evaluating constraints in orders different from the previous monotonic approaches. System designers can decide the order according to the system features.

3.5 Runtime Monitor Design

Based on the satisfaction boundary computed above, we design a runtime monitor to verify whether the environment satisfies each $W(m, k)$ in $C(K)$. Depending on the satisfaction boundary, we can then determine whether a property P can be guaranteed. If P cannot be guaranteed, we can switch the system to a safe mode. As shown in Algorithm 4, the runtime monitor only needs to store the satisfaction boundary $B[]$, instead of the safety table, in advance, reducing the space complexity from $O(K^2)$ to $O(K)$.

Besides the satisfaction boundary, the runtime monitor only needs two additional arrays, $I[k]$ for the last k-th inputs and $N_1[k]$ for the number of 1's among the last k inputs, where $1 \leq k \leq K$. During runtime (Lines 7–17), the runtime monitor reads an input (Line 8) and, for each k (Line 9), it updates the number of 1's among the last k inputs, $N_1[k]$ (Line 10), and checks if it exceeds the satisfaction boundary $B[k]$ (Line 11). If yes, it means that P is not guaranteed to be satisfied, and the system switches to a safe mode (Line 12). The runtime monitor then stores the input (Line 15) and continues monitoring.

4 DISCUSSION ON OPTIMAL APPROACHES

In this section, we define optimal approaches. The main purposes are to appraise the efficiency of the proposed approaches in Section 3 (by checking if any evaluation of weakly-hard constraints is actually not necessary) and demonstrate that there exists no deterministic algorithm that computes

an optimal verified set without being given a satisfaction boundary as an input, *i.e.*, an optimal approach needs to know the satisfaction boundary in advance. It should be emphasized that an optimal approach cannot be applied to solve the problem defined in Section 2 where the satisfaction boundary is not given.

4.1 Definitions

Definition 6. ***Verified Set***. *Given a system, a property, and an approach, the verified set of the approach is the set of weakly-hard constraints verified (not by implications) by the approach to compute the satisfaction boundary.*

Definition 7. ***Implied Set***. *Given a system, a property, and an approach, the implied set of the approach is the set of weakly-hard constraints implied by the weakly-hard constraints in the verified set.*

Based on the definitions, the union of the verified set and the implied set is $C(K)$. Assuming that the verification cost for each weakly-hard constraint can be 1, its complexity, or its runtime, we can define an optimal approach and an optimal verified set as follows:

Definition 8. ***Optimal Approach***. *For any system and any property, an optimal approach computes the satisfaction boundary and minimizes the total verification cost.*

Definition 9. ***Optimal Verified Set***. *Given a system and a property, the verified set of an optimal approach is an optimal verified set.*

Theorem 5. *There exists no optimal approach without being given a satisfaction boundary as an input.*

Proof. Given $K \in \mathbb{Z}^+$, for any (deterministic) approach without being given a satisfaction boundary, the first verified $W(m,k)$ is always the same. Since there is no single $W(m,k)$ that appears in all the optimal verified sets after we enumerate all possible satisfaction boundaries, the first verified $W(m,k)$ is not in the optimal verified sets of some systems and some properties. Therefore, there exists no optimal approach without being given a satisfaction boundary as an input. □

Consider an example with $K = 8$ and assume that the verified set of an approach is $\{W(1,2), W(1,3), W(2,5), W(3,7), W(3,8)\}$ and the optimal verified set is $\{W(1,2), W(2,5), W(3,7), W(3,8)\}$, where the given property P is satisfied under $W(3,8)$. By Theorems 2 and 3, $W(1,3) \succ W(3,8)$, and thus P is satisfied under $W(1,3)$, which does not need to be evaluated. Therefore, $W(1,3)$ is not included in the optimal verified set. However, the approach does not know the satisfaction boundary in advance, so evaluating $W(3,8)$ first does not make it an optimal approach — if P is unsatisfied under $W(3,8)$, $W(1,3)$ still needs to be evaluated. Furthermore, if P is unsatisfied under $W(1,3)$, evaluating $W(3,8)$ is a waste as it can be implied by $W(1,3)$.

4.2 Optimal Verified Set Computation

Given the satisfaction boundary B, we propose Algorithm 5 to compute an optimal verified set, as Definition 9. We first initialize a 2-dimensional array I (Lines 2–7). $I[m'][k'] \leftarrow W(m,k)$ means that $W(m',k')$ can be implied by $W(m,k)$, *i.e.*, either $W(m',k') \succ W(m,k)$ or $W(m,k) \succ W(m',k')$, which depends on the satisfaction boundary B. For example, if a property P is satisfied under $W(m,k)$, $I[m'][k'] \leftarrow W(m,k)$ means $W(m',k') \succ W(m,k)$; otherwise, if a property P is unsatisfied under $W(m,k)$, $I[m'][k'] \leftarrow W(m,k)$ means $W(m,k) \succ W(m',k')$. Then, we can iteratively update I (Lines 8–14) by B and Implications 1, 2, 3, and 4. For each $W(m,k)$, we can find

Algorithm 5 Optimal Verified Set Computation

```
1:  procedure Compute_Optimal_Verified_Set(K, B[])
2:      I ← [ ][ ]
3:      for k ← 1 to K do
4:          for m ← 1 to k do
5:              I[m][k] ← W(m, k)                          ▷ Initialize array I
6:          end for
7:      end for
8:      for k ← 1 to K do
9:          for m ← 1 to k do
10:             for W(m', k') implied by W(m, k) do        ▷ Use B and Implications in Section 3
11:                 I[m'][k'] ← I[m][k]
12:             end for
13:         end for
14:     end for
15:     return the set of W(m, k) where I[m][k] = W(m, k)
16: end procedure
```

a set of $W(m', k')$ which can be implied by $W(m, k)$ (Lines 10–12) since the satisfaction boundary is given. After that, an optimal verified set is a set of $W(m, k)$ where $I[m][k] = W(m, k)$ (Line 15). The size of the optimal verified set is at most K. It should also be mentioned that Algorithm 5 is applicable to any verification cost (*e.g.,* 1, its complexity, or its runtime) for each weakly-hard constraint. This is because a weakly-hard constraint is in the optimal verified set if and only if it cannot be implied by any other constraint in $C(K)$ — this is not affected by the definition of a verification cost.

4.3 Correctness and Uniqueness

We will prove that Algorithm 5 outputs an optimal verified set, and the optimal verified set is unique. We will demonstrate that any weakly-hard constraint in the optimal verified set cannot be implied by any other constraint in $C(K)$. To complete the proof, we provide the following definitions first.

Definition 10. ***Trace Set***. *The trace set of a weakly-hard constraint $W(m, k)$ is defined as $S(W(m, k)) = \{\sigma \mid \sigma \models W(m, k)\}$.*

Definition 11. ***Equivalence of Weakly-Hard Constraints***. *Given two weakly-hard constraints $W(m, k)$ and $W(m', k')$, we define that $W(m, k)$ is equivalent to $W(m', k')$, denoted as $W(m, k) = W(m', k')$, if and only if $S(W(m, k)) = S(W(m', k'))$.*

Theorem 6. *For any $m, m', k, k' \in \mathbb{Z}^+, m < k, m' < k'$, if $W(m, k) = W(m', k')$, then $m = m'$ and $k = k'$.*

Proof. If $m \neq m'$ or $k \neq k'$, then there is a trace σ such that either "$\sigma \models W(m, k)$ and $\sigma \not\models W(m', k')$" or "$\sigma \models W(m', k')$ and $\sigma \not\models W(m, k)$," where σ can be set as follows: if $m \neq m'$, then $\sigma = 1^{\max(m,m')}$ so that $\sigma \models W(\max(m, m'), k)$ and $\sigma \not\models W(\min(m, m'), k')$; if $m = m'$ and $k \neq k'$, then $\sigma = 1^m 0^{\min(k,k')-m} 1$ so that $\sigma \models W(m, \min(k, k'))$ and $\sigma \not\models W(m, \max(k, k'))$. By contraposition, if $W(m, k) = W(m', k')$, then $m = m'$ and $k = k'$. □

Definition 12. ***Comparability of Weakly-Hard Constraints***. *Given two weakly-hard constraints $W(m, k)$ and $W(m', k')$, we define that $W(m, k)$ and $W(m', k')$ are comparable if and only*

if either $W(m,k) \succ W(m',k')$, $W(m',k') \succ W(m,k)$, or $W(m,k) = W(m',k')$; otherwise, we define that $W(m,k)$ and $W(m',k')$ are incomparable.

4.3.1 Other Single-Constraint Implications. Here, we prove that, for any pair of weakly-hard constraints, the implication between them is covered by Theorems 1, 2, 3, and 4. As a result, we do not need to consider other implications by single weakly-hard constraints.

Theorem 7. *For any $m, m', k, k' \in \mathbb{Z}^+$, $m < k$, $m < m'$, $m \nmid m'$, $\left\lfloor \frac{m'}{m} \right\rfloor \cdot k + m' - \left\lfloor \frac{m'}{m} \right\rfloor \cdot m < k'$, $W(m,k)$ and $W(m',k')$ are incomparable.*

Proof. We first prove that $W(m,k) \succ W(m',k')$ is false. Let $k^* = \left\lfloor \frac{m'}{m} \right\rfloor \cdot k + m' - \left\lfloor \frac{m'}{m} \right\rfloor \cdot m + 1$ and $\sigma = (1^m 0^{k-m})^{\left\lfloor \frac{m'}{m} \right\rfloor} 1^{k^* - \left\lfloor \frac{m'}{m} \right\rfloor \cdot k}$. It is trivial that $\sigma \models W(m,k)$, but $\sigma \not\models W(m',k^*)$ because σ has its length k^* and $(m'+1)$ 1's. Therefore, for $k' \geq k^*$, $\sigma \not\models W(m',k')$. We then prove that $W(m',k') \succ W(m,k)$ is false. Let $\sigma = 1^{m'} 0^{k'-m'}$. It is trivial that $\sigma \models W(m',k')$, but $\sigma \not\models W(m,k)$ because there are more than m 1's in the first k inputs. Combining the two proofs, $W(m,k)$ and $W(m',k')$ are incomparable. □

Theorem 8. *For any $m, m', k, k' \in \mathbb{Z}^+$, $m < k$, $m < m'$, $m \mid m'$, $\frac{m'}{m} \cdot k < k'$, $W(m,k)$ and $W(m',k')$ are incomparable.*

Proof. We first prove that $W(m,k) \succ W(m',k')$ is false. Let $k^* = \frac{m'}{m} \cdot k + 1$ and $\sigma = (1^m 0^{k-m})^{\frac{m'}{m}} 1$. It is trivial that $\sigma \models W(m,k)$, but $\sigma \not\models W(m',k^*)$ because σ has its length k^* and $(m'+1)$ 1's. Therefore, for $k' \geq k^*$, $\sigma \not\models W(m',k')$. We then prove that $W(m',k') \succ W(m,k)$ is false. Let $\sigma = 1^{m'} 0^{k'-m'}$. It is trivial that $\sigma \models W(m',k')$, but $\sigma \not\models W(m,k)$ because there are more than m 1's in the first k inputs. Combining the two proofs, $W(m,k)$ and $W(m',k')$ are incomparable. □

Theorems 7 and 8 cover all cases when two weakly-hard constraints are incomparable.

Theorem 9. *For any $m_1, m_2, m_3, k_1, k_2, k_3 \in \mathbb{Z}^+, m_1 \leq k_1, m_2 \leq k_2, m_3 \leq k_3$, if $W(m_1,k_1) \succ W(m_2,k_2)$ and $W(m_2,k_2) \succ W(m_3,k_3)$, then $W(m_1,k_1) \succ W(m_3,k_3)$.*

Proof. By Definition 4, any input trace that satisfies $W(m_1,k_1)$ also satisfies $W(m_2,k_2)$, and any input trace that satisfies $W(m_2,k_2)$ also satisfies $W(m_3,k_3)$. Therefore, any input trace that satisfies $W(m_1,k_1)$ also satisfies $W(m_3,k_3)$. □

Theorem 9 is the transitive law for the comparability of weakly-hard constraints, and we can combine the theorems to get more implications.

Theorem 10. *For any $m, m', k, k' \in \mathbb{Z}^+$, $m < k$, $m' < k'$, $(m,k) \neq (m',k')$, $W(m,k)$ and $W(m',k')$ are either incomparable or comparable and implied by the combination of Theorems 1, 2, 3, and 4.*

Proof. Given an $W(m,k)$, we define $\Gamma = \{W(m',k') | (m,k) \neq (m',k'), m' \geq m\}$. Any $W(m',k') \in \Gamma$ is corresponding to one of the following cases:

- If $k' = k$, then $W(m,k) \succ W(m',k')$ or $W(m',k') \succ W(m,k)$ by Theorem 1.
- If $m' = m$, then $W(m,k) \succ W(m',k')$ or $W(m',k') \succ W(m,k)$ by Theorem 2.
- If $m' > m, m \nmid m', k' \leq \left\lfloor \frac{m'}{m} \right\rfloor \cdot k + m' - \left\lfloor \frac{m'}{m} \right\rfloor \cdot m$, then $W(m,k) \succ W(m',k')$ by the combination of Theorems 2, 3, and 4.
- If $m' > m, m \nmid m', k' > \left\lfloor \frac{m'}{m} \right\rfloor \cdot k + m' - \left\lfloor \frac{m'}{m} \right\rfloor \cdot m$, then $W(m,k)$ and $W(m',k')$ are incomparable by Theorem 7.
- If $m' > m, m \mid m', k' \leq \frac{m'}{m} \cdot k$, then $W(m,k) \succ W(m',k')$ by the combination of Theorems 2 and 3.
- If $m' > m, m \mid m', k' > \frac{m'}{m} \cdot k$, then $W(m,k)$ and $W(m',k')$ are incomparable by Theorem 8.

□

Theorem 10 shows that Theorems 1, 2, 3, 4, 7, and 8 cover all possible cases for a pair of weakly-hard constraints. However, Theorems 7 and 8 indicate that the weakly-hard constraints are incomparable. As a result, we only need to consider Theorems 1, 2, 3, and 4 for the implications by a single weakly-hard constraint.

4.3.2 Multiple-Constraint Implications. Here, we prove that, if the combination of n weakly-hard constraints $\{W(m_1,k_1), W(m_2,k_2), \ldots, W(m_n,k_n)\}$ implies another weakly-hard constraint $W(m,k)$, then a weakly-hard constraint $W(m_i,k_i) \in \{W(m_1,k_1), W(m_2,k_2), \ldots, W(m_n,k_n)\}$ implies $W(m,k)$. As a result, we do not need to consider the implications by multiple weakly-hard constraints.

Theorem 11. *If there is a set of n weakly-hard constraints $\{W(m_1,k_1), W(m_2,k_2), \ldots, W(m_n,k_n)\}$ and another weakly-hard constraint $W(m,k)$ such that $S(W(m,k)) \subseteq \bigcup_{i=1}^{n} S(W(m_i,k_i))$, then there must be a weakly-hard constraint $W(m_i,k_i) \in \{W(m_1,k_1), W(m_2,k_2), \ldots, W(m_n,k_n)\}$ such that $W(m,k) \succ W(m_i,k_i)$.*

Proof. Let $\sigma = (1^m 0^{k-m})^n$ and $\Gamma = \{W(m',k') | m' < k'\}$. Any $W(m',k') \in \Gamma$ is corresponding to one of the following cases:

- If $m' < m$, then $\sigma \not\models W(m',k')$ since $m' < m$.
- If $m' = m, k' > k$, then $\sigma \not\models W(m',k')$ since the $(k+1)$-th element in σ is 1.
- If $W(m,k)$ and $W(m',k')$ are incomparable, then $\sigma \not\models W(m',k')$ as σ is used in proving Theorems 7 and 8.
- For another other $W(m',k')$, the proof in Theorem 10 states that $W(m,k) \succ W(m',k')$ and thus $\sigma \models W(m',k')$.

Considering all cases, if $\sigma \models W(m',k')$, then $W(m,k) \succ W(m',k')$. Therefore, Theorem 11 is proved. □

Theorem 11 indicates that we do not need to consider the implications by multiple weakly-hard constraints. If $S(W(m,k)) \subseteq \bigcup_{i=1}^{n} S(W(m_i,k_i))$ and P is satisfied under each $W(m_i,k_i)$, then P is satisfied under $W(m,k)$. By Theorem 11, the implication (P is satisfied under $W(m,k)$) can actually be obtained from a single-constraint implication.

On the other hand, another implication is that, if $S(W(m,k)) \supseteq \bigcup_{i=1}^{n} S(W(m_i,k_i))$ and P is unsatisfied under at least one $W(m_i,k_i)$, then P is unsatisfied under $W(m,k)$. This implication can also be obtained from a single-constraint implication (from a constraint $W(m_i,k_i)$ making P unsatisfied).

4.3.3 Completion of Proof.

Theorem 12. *Algorithm 5 outputs an optimal verified set, and the optimal verified set is unique.*

Proof. By Theorem 11 and the explanation above, a multiple-constraint implication can be obtained from a single-constraint implication. By Theorem 10, a single-constraint implication between constraints which are comparable is covered by Theorems 1, 2, 3, and 4. Therefore, there is no other implication, and Algorithm 5 outputs the unique optimal verified set. □

It should be mentioned that there are many optimal paths, and each of them includes the same set of (m,k) (where $I[m][k] = W(m,k)$) with different sequences (orders), as returned by Algorithm 5.

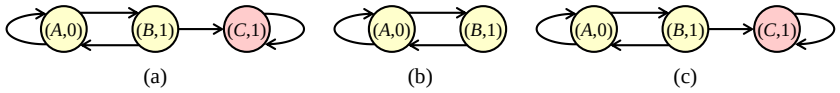

Fig. 4. The mask-compressing approach constructs the graphs from the example in Figure 1 for (a) $W(1,1)$, (b) $W(1,2)$, and (c) $W(2,2)$.

5 REACHABILITY ANALYSIS FOR FINITE-STATE MACHINES

In this section, we consider a special case of system verification with weakly-hard constraints — reachability analysis for finite-state machines. We first propose a mask-compressing approach (Section 5.2) to verify reachability under a single weakly-hard constraint . The mask-compressing approach serves as the example of verifying a property P (reachability) under a single constraint in $C(K)$, and thus it can be plugged into (called by) the approaches in Section 3. Then, we propose a layered BFS approach (Section 5.3) which computes the safety table in a more efficient way — the layered BFS approach computes the safety table with the same complexity as evaluating a single (m, K) constraint (K is the upper bound of all possible values of k). We also propose a dual-layered BFS approach (Section 5.4) which has the same theoretical complexity but reduces the number of BFS traverses.

5.1 Problem Definition

A non-deterministic finite-state machine model S is defined as $\langle Q, \Sigma, \delta, P_r, q_0, F\rangle$ where Q is the finite set of states, $\Sigma = \{0, 1\}$ is the set of input symbols, $\delta \subseteq Q \times \Sigma \times Q$ is the transition table, $P_r : \delta \longrightarrow (0, 1]$ is the transition probability satisfying

$$\forall (q, x) \in Q \times \Sigma, \sum_{\overline{q} \in Q, (q,x,\overline{q}) \in \delta} P_r(q, x, \overline{q}) = 1, \tag{2}$$

where q_0 is the initial state, and $F \subseteq Q$ is the finite set of unsafe states. Given a finite-state machine S and a positive integer K, the goal is to determine whether the property P of "never reaching an unsafe state" is satisfied with all possible traces under each $W(m, k)$ in $C(K)$.

5.2 Mask-Compressing Approach

We develop the masking-compressing approach to verify the reachability property P under a single weakly-hard constraint $W(m, k)$. Again, it should be emphasized that the mask-compressing approach serves as the example of verifying a property P (reachability) under a single constraint in $C(K)$, and thus it can be plugged into (called by) the approaches in Section 3. The mask-compressing approach traverses a finite-state machine with all possible traces that satisfy the weakly-hard constraint. It records the previous $k-1$ inputs and considers the possibility of the next input. Since there are at most m 1's among any k consecutive inputs, if there have been m 1's among previous $k-1$ inputs, then the next input must be 0.

Given the previous $k-1$ inputs, we encode them by compressing them into a $(k-1)$-bit mask. Formally, given a finite state machine $S = \langle Q, \Sigma, \delta, P_r, q_0, F\rangle$, we define a graph to perform verification for a single weakly-hard constraint $W(m, k)$ as follows:

- The vertex set is the set product of the states of S and the $(k-1)$-bit mask.
- There is a directed edge from $v_{q,mask}$ to $v_{\overline{q},\overline{mask}}$ if and only if

$$(q,\ \overline{mask}\ \%\ 2,\ \overline{q}) \in \delta, \tag{3}$$

$$(mask \cdot 2)\ \%\ 2^{k-1} + \overline{mask}\ \%\ 2 = \overline{mask}, \tag{4}$$

$$\text{Count1}(mask) + \overline{mask} \,\%\, 2 \leq m, \tag{5}$$

where q and $\overline{q}$ are states, $mask$ and $\overline{mask}$ are $(k-1)$-bit masks ($(q, mask)$ or $(\overline{q}, \overline{mask})$ can be regarded as the index of a vertex), and Count1() counts the number of 1's in a mask.

Equation (3) is for the transition in S, Equation (4) is for the 1-bit "shift" of the mask, and Equation (5) is for the number of 1's bounded by the weakly-hard fault model. An example is shown in Figure 4. The mask-compressing approach constructs the graphs from the example in Figure 1 for $W(1,1)$, $W(1,2)$, and $W(2,2)$, respectively. After constructing the graph, we can traverse the graph with a BFS starting from $v_{q_0,0}$, and P is unsatisfied if and only if we can reach a vertex $v_{q,mask}$ where $q \in F$. Note that this is equivalent to verifying the composition of S and the state machine representing a single weakly-hard constraint $W(m,k)$. Here, we use the mask-compressing approach because the bit operations not only support more efficient computation but also consumes less memory.

The graph has at most $|Q| \cdot 2^k$ vertices and $|\delta| \cdot 2^k$ edges, and thus the complexity is $O(N \cdot 2^k)$, where $N = |Q| + |\delta|$, for the mask-compressing approach verifying the reachability property P under a single $W(m,k)$. The complexity is optimal to the problem because we always need to keep track of the previous $k-1$ inputs and the current input (implying 2^k possible cases) so that we can update the number of 1's (among k consecutive events) once there is a new input. When plugging the masking-compressing approach into the approaches in Section 3, the complexities are as follows:

- Algorithm 1: $O\left(\sum_{k=1}^{K} 2 \cdot N \cdot 2^k\right) = O\left(N \cdot 2^{K+1} - N \cdot 2\right) = O(N \cdot 2^K)$.
- Algorithm 2: $O\left(\sum_{k=1}^{K} N \cdot 2^k\right) = O\left(N \cdot 2^{K+1} - N \cdot 2\right) = O(N \cdot 2^K)$.
- Algorithm 3: it depends on the cost estimation and constraint implication.

5.3 Layered BFS Approach

Here, we propose the layered BFS approach which computes the safety table in a more efficient way. The key insight of the layered BFS approach is that multiple weakly-hard constraints $W(m,k)$ with the same k can be verified together within a BFS.

Theorem 13. *For $W(m,k)$, $W(m+1,k) \in C(K)$, the graph for $W(m,k)$ constructed by the mask-compressing approach is a subgraph of the graph for $W(m+1,k)$.*

Proof. By Equation (5), if an edge is in the graph for $W(m,k)$, it must also be in the graph for $W(m+1,k)$. □

Theorem 14. *Each reachable vertex in the graph for $W(m+1,k)$ is also reachable from the initial states of the graph for $W(m,k)$.*

Proof. It is straightforward by Theorem 13. Note that the initial vertices for the graphs for $W(m,k)$ and $W(m+1,k)$ are the same. □

Theorem 13 implies that evaluating $W(m,k)$ leads to the results for all $W(m',k)$, where $1 \leq m' \leq m$. Thus, only the graph for $W(k,k)$ needs to be traversed for all $W(m',k)$, where $1 \leq m' \leq k$. Theorem 14 further implies that we can perform BFS for k iterations from the graph for $W(1,k)$ to the graph for $W(k,k)$, called the "layered BFS approach" in this paper. Formally, we denote the sets of edges and vertices in the graph for $W(m,k)$ as E_m and V_m respectively. For the m-th iteration (as a layer), we perform BFS on the graph $G_m = (V_m, E_m)$. We exploit the previous result of the BFS on $G_{m-1} = (V_{m-1}, E_{m-1})$ and thus avoid redundancy as $G_{m-1} \subseteq G_m$.

An example is shown in Figure 5(a), where vertices v_1 and v_2 are reachable (satisfying Equation (3)) and other vertices are unreachable (not satisfying Equation (3)). After performing the BFS for $W(m,k)$, we can collect a vertex set V'_m:

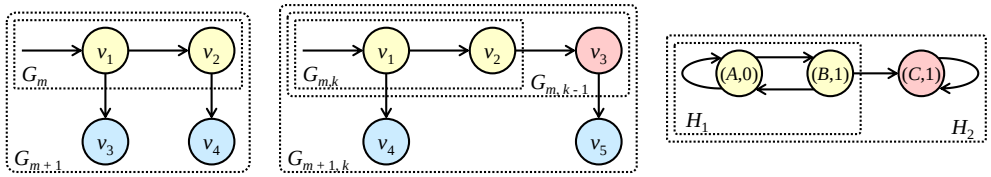

Fig. 5. (a) An example layered BFS from $W(m,k)$ to $W(m+1,k)$. Vertices v_1 and v_2 are reachable under $W(m,k)$. Vertices v_3 and v_4 are unreachable under $W(m,k)$ but reachable under $W(m+1,k)$. (b) An example dual-layered BFS from $W(m,k)$ to $W(m,k-1)$ and then to $W(m+1,k)$. Vertices v_1 and v_2 are reachable under $W(m,k)$. Vertex v_3 is unreachable under $W(m,k)$ but reachable under $W(m,k-1)$. Vertices v_4 and v_5 are unreachable under $W(m,k)$ and $W(m,k-1)$ but reachable under $W(m+1,k)$. (c) The graphs from the example in Figure 1, where the small graph H_1 is G_1 of the layered BFS ($k=2$) as well as $G_{1,2}$ for the dual-layered BFS, and the large graph H_2 is G_2 of the layered BFS ($k=2$) as well as $G_{1,1} = G_{2,2}$ for the dual-layered BFS. Note that the dual-layered BFS just needs one traverse for $k=1$ and $k=2$, but the layered BFS needs one traverse for each of $k=1$ and $k=2$.

- For each $v' \in V'_m$, there exists a vertex $v \in V_m$ such that $(v,v') \in E_{m+1}$.

In the example, $V'_m = \{v_3, v_4\}$ and the corresponding edges in E_{m+1} are (v_1, v_3) and (v_2, v_4), respectively. By Theorems 13 and 14, after starting from the vertices in V'_m and performing the BFS on G_{m+1}, we traverse all vertices in V_{m+1} without repeating the BFS on G_m. Note that the mask of each vertex in V'_m satisfies $W(m+1,k)$. After the iterations from $W(m,k)$ to $W(k,k)$, each vertex in G_k is traversed only once. Moreover, if an unsafe state is reached in the m-th iteration, P is only guaranteed to be satisfied under $W(m',k)$, where $m' < m$. The corresponding graphs from the example in Figure 1 are shown in Figure 5(c).

Since each vertex in the graph for $W(k,k)$ only needs to be traversed once, the complexity for a given k is $O(N \cdot 2^k)$, where $N = |Q| + |\delta|$. The total complexity for all k is $O\left(\sum_{k=1}^{K} N \cdot 2^k\right) = O\left(N \cdot 2^{K+1} - N \cdot 2\right) = O(N \cdot 2^K)$. This shows that the layered BFS approach computes the satisfaction boundary with the same complexity as Algorithms 1 and 2 as well as verifying a single (m,K) constraint (K is the upper bound of all possible values of k).

5.4 Dual-Layered BFS Approach

In the layered BFS approach, the graph for $W(m,k)$ is constructed by the mask-compressing approach with a $(k-1)$-bit mask, whereas the graph for $W(m,k-1)$ is constructed with a $(k-2)$-bit mask. As a result, the same input trace is encoded into different vertices and edges in the two graphs, and thus it requires two traversals to perform verification. Here we propose to construct both graphs with a $(k-1)$-bit mask so that the weakly-hard constraints $W(m,k)$ and $W(m,k-1)$ can be verified within a BFS.

Theorem 15. *For $W(m,k), W(m,k-1) \in C(K)$, the graph for $W(m,k)$ constructed by the mask-compressing approach with a $(k-1)$-bit mask is a subgraph of the graph for $W(m,k-1)$, also constructed with a $(k-1)$-bit mask.*

Theorem 16. *For $W(m,k-1), W(m+1,k) \in C(K)$, the graph for $W(m,k-1)$ constructed by the mask-compressing approach with a $(k-1)$-bit mask is a subgraph of the graph for $W(m+1,k)$, also constructed with a $(k-1)$-bit mask.*

Theorem 15 implies that evaluating $W(m,k-1)$ leads to the result of $W(m,k)$, and Theorem 16 implies that evaluating $W(m+1,k)$ leads to the result of $W(m,k-1)$. Therefore, a single BFS on

the graph for $W(k,k)$ allows us to compute the satisfaction boundaries $B(k-1)$ and $B(k)$. Formally, we denote the sets of edges and vertices in the graph for $W(m,k)$ as $E_{m,k}$ and $V_{m,k}$ respectively. Upon performing the BFS on $G_{m,k-1} = (E_{m,k-1}, V_{m,k-1})$, we exploit the previous result of BFS on $G_{m,k} = (E_{m,k}, V_{m,k})$ and avoid redundancy as $G_{m,k} \subseteq G_{m,k-1}$. Similarly, upon performing the BFS on $G_{m+1,k} = (E_{m+1,k}, V_{m+1,k})$, we exploit the previous result of BFS on $G_{m,k-1}$ and avoid redundancy as $G_{m,k-1} \subseteq G_{m+1,k}$. Note that all these graphs are constructed with a $(k-1)$-bit mask.

An example is shown in Figure 5(b), where vertices v_1 and v_2 are reachable under $W(m,k)$, vertex v_3 is unreachable under $W(m,k)$ but reachable under $W(m,k-1)$, and vertices v_4 and v_5 are unreachable under $W(m,k)$ and $W(m,k-1)$ but reachable under $W(m+1,k)$. During the BFS on $G_{m,k}$, we collect two vertex sets $V'_{m,k-1}$ and $V'_{m+1,k}$:

- For each $v' \in V'_{m,k-1}$, there exists a vertex $v \in V_{m,k}$ such that $(v,v') \notin E_{m,k}$ and $(v,v') \in E_{m,k-1}$.
- For each $v' \in V'_{m+1,k}$, there exists a vertex $v \in V_{m,k}$ such that $(v,v') \notin E_{m,k-1}$ (thus $(v,v') \notin E_{m,k}$) and $(v,v') \in E_{m+1,k}$.

In the example, $V'_{m,k-1} = \{v_3\}$ and $V'_{m+1,k} = \{v_4\}$. After the BFS on $G_{m,k}$ and upon the BFS on $G_{m,k-1}$, we start from the vertices in $V'_{m,k-1}$ and avoid redundant traversal of $G_{m,k}$. Similarly, during the BFS on $G_{m,k-1}$, we collect a vertex set $V''_{m+1,k}$:

- For each $v'' \in V''_{m+1,k}$, there exists a vertex $v \in V_{m,k-1} \setminus V_{m,k}$ such that $(v,v'') \notin E_{m,k-1}$ and $(v,v'') \in E_{m+1,k}$.

In the example, $V''_{m+1,k} = \{v_5\}$. After the BFS on $G_{m,k-1}$ and upon the BFS on $G_{m+1,k}$, we start from the vertices in $V'_{m+1,k} \cup V''_{m+1,k}$ and avoid redundant traversal of $G_{m,k-1}$. As a result, every vertex in $G_{k,k}$ is traversed at most once in order to compute $B(k-1)$ and $B(k)$, and thus half of the BFS can be reduced. The corresponding graphs from the example in Figure 1 are shown in Figure 5(c). Note that the dual-layered BFS approach just needs one traverse for $k=1$ and $k=2$, but the layered BFS approach needs one traverse for each of $k=1$ and $k=2$.

If an unsafe state $q \in F$ is reached during the BFS on $G_{m,k}$, it is clear that $B(k) = m-1$. By Implication 2, $B(k-1) = m-1$. On the other hand, if an unsafe state $q \in F$ is reached during the BFS on $G_{m,k-1}$, it is clear that the $B(k-1) = m-1$. Since $G_{m,k}$ has already been traversed without reaching any unsafe states, by Implication 4, $B(k) = m$.

The total complexity for all k is $O\left(\frac{\sum_{k=1}^{K} N \cdot 2^k}{2}\right) = O\left(N \cdot 2^K - N\right) = O(N \cdot 2^K)$. It is the same as the complexity of the layered BFS approach, but the dual-layered BFS approach is more efficient as it can reduce half of the BFS.

Note that the graphs constructed by the mask-compressing approach, the layered BFS approach, and the dual-layered BFS approach can be used to store which transitions (or edges) are constrained (or blocked) by weakly-hard constraints. However, for complicated properties, *e.g.*, nested Linear Temporal Logic (LTL) properties, the complexity may be much higher. The main idea of the layered BFS approach and the dual-layered BFS approach is that we do not need to re-traverse a vertex if the vertex has been traversed, but re-traversing may be needed for complicated properties.

6 EXPERIMENTAL RESULTS

To compare the efficiency of different approaches, we implemented a brute-force approach that evaluates all constraints in $C(K)$ one by one (BF), the monotonic approach (MONO, Algorithm 1), the monotonic approach with dynamic upper bound of satisfaction boundary (MONO-DUB, Algorithm 2), the lowest-cost-first heuristic (LCF, Algorithm 3), which defines the estimated cost for evaluating $W(m,k)$ as $\sum_{i=0}^{m} \binom{k-1}{i}$, the optimal approach (OPT, Algorithm 5), the layered BFS approach (L-BFS), and the dual-layered BFS approach (DL-BFS). Except the optimal approach, the

layered BFS approach, and the dual-layered BFS approach, the other four approaches call the mask-compressing approach when they need to evaluate a single constraint in $C(K)$. The approaches are implemented in C++ (for verification) and Python (for input generation and runtime computation)[2]. The whole experiments are run on a Linux desktop with the Intel Core i7-9700 processor and 16GB memory.

6.1 Discrete Second-Order Control

6.1.1 Setting. The case study is a discrete second-order controller under perturbation attacks.The following parameters are in meter, second, or their combinations. We denote the control value, its first-order derivative, and its second-order derivative at time t as $x(t)$, $\dot{x}(t)$, and $\ddot{x}(t)$, respectively. The objective of the controller is to maintain x at a fixed value (0 in our case), and the attacker attempts to shift x away from the fixed value. A control configuration is formally defined as $\langle x_{\min}, x_{\max}, \dot{x}_{\min}, \dot{x}_{\max}, \ddot{x}_C, S_{\text{atk}} \rangle$, where

- $[x_{\min}, x_{\max}]$ is the safe range. If x exceeds the range, the safety property is violated.
- $[\dot{x}_{\min}, \dot{x}_{\max}]$ is the physical constraint for the first order derivative of x. If the controller attempts to set $\dot{x}$ to a value larger (smaller) than $\dot{x}_{\max}$ ($\dot{x}_{\min}$), $\dot{x}$ is set to the corresponding limit.
- $\ddot{x}_C$ is the constant magnitude for the second order derivative of x, *i.e.*, $\ddot{x}(t) \in \{-\ddot{x}_C, 0, \ddot{x}_C\}$.
- S_{atk} is the set of possible attack values on x.

Suppose the control value x deviates away from 0, the policy of the controller is to accelerate until $\dot{x}$ reaches the limit $(\dot{x}_{\min}, \dot{x}_{\max})$ and decelerate when the control value x is approaching 0. The timing to start the deceleration is determined such that $\dot{x} = 0$ when $x = 0$, and we denote the value of x at which the deceleration starts as x_{dec}, which is

$$x_{\text{dec}}(t) = \dot{x}(t) \cdot t_{\text{dec}}(t) - \frac{1}{2} \cdot \text{sign}(\dot{x}(t)) \cdot \ddot{x}_C \cdot t_{\text{dec}}(t)^2, \tag{6}$$

where $t_{\text{dec}}(t) = \frac{|\dot{x}(t)|}{\ddot{x}_C}$ is the time required to decelerate $\dot{x}(t)$ to 0. The transition functions of the controller can be expressed as

$$x(t+1) \leftarrow x(t) + \dot{x}(t) + p_{\text{atk}}(t), \tag{7}$$

$$\dot{x}(t+1) \leftarrow \max\left(\min\left(\dot{x}(t) + \ddot{x}(t), \dot{x}_{\max} \right), \dot{x}_{\min} \right), \tag{8}$$

$$\ddot{x}(t+1) \leftarrow -\text{sign}\Big(x(t) + p_{\text{atk}}(t) \Big) \cdot \text{sign}\Big(|x(t) + p_{\text{atk}}(t)| - |x_{\text{dec}}(t)| \Big) \cdot \ddot{x}_C, \tag{9}$$

where p_{atk} denotes the perturbation attack. Equation (7) is for the transition of x, where the control value is affected by both the first-order derivative and the perturbation attack. Equation (8) is for the transition of $\dot{x}$, with the updated value clipped to $[\dot{x}_{\min}, \dot{x}_{\max}]$ to satisfy the physical constrain. Equation (9) is for the transition of $\ddot{x}$, where the sign of $\ddot{x}$ is determined by the relative position of x with respect to 0 and whether the system is decelerating as x approaches 0. The safety property is $x \in [x_{\min}, x_{\max}]$, meaning that the control value stays in the safe range.

For any controller configuration $\langle x_{\min}, x_{\max}, \dot{x}_{\min}, \dot{x}_{\max}, \ddot{x}_C, S_{\text{atk}} \rangle$, we can define a finite state machine $\langle Q, \Sigma, \delta, P_r, q_0, F \rangle$, where

- $Q = \{(x, \dot{x}, \ddot{x}) | x, \dot{x} \in \mathbb{Z}, x \in [x_{\min}, x_{\max}], \dot{x} \in [\dot{x}_{\min}, \dot{x}_{\max}], \ddot{x} \in \{-\ddot{x}_C, 0, \ddot{x}_C\}\} \cup \{q_{\text{unsafe}}\}$.
- $\Sigma = S_{\text{atk}} \cup \{0\}$.
- δ is defined exactly from the transition functions above.
- $P_r((x, \dot{x}, \ddot{x}), p_{\text{atk}}, (x', \dot{x}', \ddot{x}')) = \frac{1}{|S_{\text{atk}}|}$.

[2]The implementation is available at `https://gitlab.com/ntu-cps-lab/WeaklyHardVerification2022` .

Table 2. Discrete second-order control: runtime (in second) with different values of $|Q|$.

$\|Q\|$	BF	MONO	MONO-DUB	LCF	OPT	L-BFS	DL-BFS
280	2.069	0.105	0.103	0.102	0.055	0.105	0.074
314	2.506	0.146	0.144	0.144	0.071	0.147	0.099
331	9.817	1.683	1.641	1.641	1.234	1.752	1.786
341	54.730	9.624	5.128	5.128	2.655	5.621	4.434
351	61.834	11.526	5.546	5.546	3.127	6.178	4.027
361	58.244	11.098	5.039	5.039	2.802	7.088	4.450
371	63.565	12.462	5.258	5.258	2.900	7.613	4.141
381	65.523	12.995	5.229	5.229	2.900	7.999	4.329

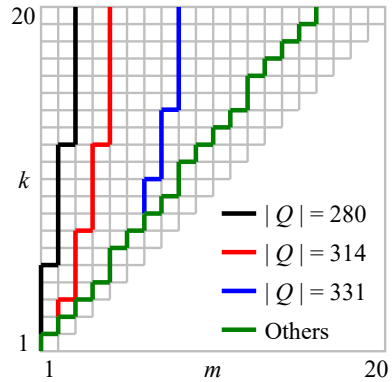

Fig. 6. Discrete second-order control: computed satisfaction boundaries.

Table 3. Discrete second-order control: runtime (in second) with different values of K.

K	BF	MONO	MONO-DUB	LCF	OPT	L-BFS	DL-BFS
14	0.388	0.062	0.057	0.057	0.032	0.061	0.041
16	1.200	0.217	0.172	0.172	0.146	0.184	0.186
18	3.047	0.373	0.333	0.333	0.177	0.356	0.289
20	9.817	1.683	1.641	1.641	1.234	1.752	1.786
22	31.041	6.596	6.555	6.555	2.931	7.023	5.297

- $q_0 = (0, 0, 0)$.
- $F = \{q_{\text{unsafe}}\}$.

q_{unsafe} represents the state where the control value x is out of the range $[x_{\min}, x_{\max}]$. Verifying whether the control value is in the safe range under perturbation attacks is reduced to solving for the reachability of q_{unsafe} for the finite-state machine.

6.1.2 Experiment on $|Q|$. We experimented on how each approach scales with respect to the number of states in the finite-state machine, $|Q|$. To create different numbers of states, we fixed $\dot{x}_{\min} = -4$, $\dot{x}_{\max} = 4$, $\ddot{x}_C = 2$, and $S_{\text{atk}} = \{5\}$ and experimented with $(x_{\min}, x_{\max}) = \{\pm 30, \pm 40, \pm 50, \ldots, \pm 100\}$, resulting $|Q|$ from 280 to 381. A larger safe range $[x_{\min}, x_{\max}]$ of the control value x allows the controller to have a larger margin to recover from attacks. K is set to 20.

The results are shown in Table 2, and the corresponding satisfaction boundaries are illustrated in Figure 6, where all approaches generate the same satisfaction boundaries. The monotonic

approach runs significantly faster than the brute-force approach because the verification results under many weakly-hard constraints are implied by Implications 1 and 2. For larger number of states, the runtime differences are even larger. We then compare the monotonic approach, the monotonic approach with dynamic upper bound of satisfaction boundary (monotonic-dynamic), and the lowest-cost-first heuristic. The results are aligned with the theoretical expectations. The monotonic-dynamic approach runs strictly faster than the monotonic approach for every setting with the addition implications by Implications 3 and 4, and the lowest-cost-first heuristic performs same as the monotonic-dynamic approach. The optimal approach finds the optimal verified set, the runtime corresponding to the the optimal verified set is smaller than the runtimes of the other approaches. However, it needs to know the satisfaction boundary in advance, so the main purpose of the optimal approach is to evaluate the efficiency of the other approaches. It can be observed that the runtime of the optimal approach may not be monotonic to $|Q|$ as different $|Q|$ lead to different boundaries and thus different optimal verified sets, where a large $|Q|$ may have a smaller optimal verified set. The layered BFS approach runs faster than the monotonic approach in most cases, and it has comparable runtime as the monotonic-dynamic approach and the lowest-cost-first heuristic. Moreover, the dual-layered BFS approach mostly has the best efficiency among all other approaches except the optimal approach.

6.1.3 Experiment on K. We experimented on how each approach scales with respect to K. We fixed $x_{\min} = -50$, $x_{\max} = 50$, $\dot{x}_{\min} = -4$, $\dot{x}_{\max} = 4$, $\ddot{x}_{\text{const}} = 2$, and $S_{\text{atk}} = \{5\}$. The results are shown in Table 3, where we report the results with $K = 14, 16, 18, 20, 22$. Similar to the previous experiment, the proposed approaches outperform the brute-force approach significantly. This is aligned with the theoretical complexity analysis that the brute-force approach needs to evaluate $O(K^2)$ weakly-hard constraints, and the other approaches need to evaluate $O(K)$ weakly-hard constraints only. It should be emphasized that the verification of a property under a single weakly-hard constraint $W(m, k)$ usually needs to store the last k inputs, and thus the complexity is at least $O(2^k)$. If the property is more complicated (*e.g.*, in Linear Temporal Logic), the complexity can be even higher. Therefore, reducing the number of evaluations of weakly-hard constraints is really advantageous to the efficiency of computing the safety table or the satisfaction boundary. It should also be mentioned that the layered BFS approach and the dual-layered BFS approach are especially for the reachability of finite-state machines, and the other proposed approaches are general and compatible with other verification approaches for a single weakly-hard constraint.

6.2 Network Routing

6.2.1 Setting. The case study is network routing of Extranet, where there are two routing paths with the same source and destinations on one router. The following parameters are in a general time unit. We denote the delay levels of two routing paths at time t as $l_1(t)$ and $l_2(t)$. We also denote the waiting times (for recovery) of two routing paths at time t as $w_1(t)$ and $w_2(t)$. The objective of the network routing is to switch between two routing paths to keep the connection between the source and the destination. A routing configuration is formally defined as $\langle \tau_1, \tau_2, \gamma \rangle$, where

- τ_1 and τ_2 are the thresholds of two routing paths, respectively. If delay level $l_i(t)$ exceed τ_i, the i-th routing path is considered to be congested, and it needs to recover.
- γ is the time (measured by the number of inputs) that a routing path needs to recover.

We introduce the following variables:

- $s(t) \in \{0, 1\}$ denotes whether the two routing paths are switched at time t.
- $c(t) \in \{0, 1\}$ denotes whether both of the two routing path are congested.

- $d(t) \in \{0, 1\}$ denotes whether a packet is delayed.

We also introduce the following transition functions:

- If $s(t) = 0$, meaning that the first routing path is in use (not switched to the second routing path), then

$$l_1(t+1) \leftarrow l_1(t) + (2 \cdot d(t) - 1); \quad w_2(t+1) \leftarrow w_2(t) + 1; \quad l_2(t+1) \leftarrow 0; \quad w_1(t+1) \leftarrow 0, \tag{10}$$

  meaning that $l_1(t+1)$ is increased or decreased by 1 from $l_1(t)$ if $d(t)$ is 1 or 0, respectively, and $w_2(t+1)$ is increased by 1 from $w_2(t)$. $l_2(t+1)$ and $w_1(t+1)$ will be set to the initial value until switching to the second routing path.
- If $s(t) = 1$, meaning that the second routing path is in use (switched from the first routing path), then

$$l_2(t+1) \leftarrow l_2(t) + (2 \cdot d(t) - 1); \quad w_1(t+1) \leftarrow w_1(t) + 1; \quad l_1(t+1) \leftarrow 0; \quad w_2(t+1) \leftarrow 0, \tag{11}$$

  meaning that $w_1(t+1)$ is increased by 1 from $w_1(t)$, and $l_2(t+1)$ is increased or decreased by 1 from $l_2(t)$ if $d(t)$ is 1 or 0, respectively. $l_1(t+1)$ and $w_2(t+1)$ will be set to the initial value until switching to the first routing path.
- The two routing paths are switched at time $t+1$ if the delay of the routing path in use at time t exceeds the corresponding threshold, and the waiting time (for recovery) of the other routing path at time t exceeds the threshold γ, *i.e.*,

$$s(t+1) \leftarrow \begin{cases} 1, & \text{if } s(t) = 0, l_1(t) > \tau_1, w_2(t) > \gamma; \\ 0, & \text{if } s(t) = 1, l_2(t) > \tau_2, w_1(t) > \gamma; \\ s(t), & \text{otherwise.} \end{cases} \tag{12}$$

- Both of the two routing path are congested if the delay of the routing path in use at time t exceeds the corresponding threshold, and the waiting time (for recovery) of the other routing path at time t does not exceed the threshold γ, *i.e.*,

$$c(t+1) \leftarrow \begin{cases} 1, & \text{if } s(t) = 0, l_1(t) > \tau_1, w_2(t) \leq \gamma; \\ 1, & \text{if } s(t) = 0, l_2(t) > \tau_2, w_1(t) \leq \gamma; \\ 0, & \text{otherwise.} \end{cases} \tag{13}$$

The safety property is $c(t) = 0$, meaning that at least one routing path is not congested.

For any network routing configuration $\langle \tau_1, \tau_2, \gamma \rangle$, we can determine a finite state machine $\langle Q, \Sigma, \delta, q_0, F \rangle$, where

- $Q = \{(l_1, l_2, w_1, w_2, s, c)\}$.
- $\Sigma = \{0, 1\}$, which is the input as $d(t)$.
- δ is defined exactly from the transition functions above.
- $q_0 = (0, 0, 0, 0, 0, 0)$.
- $F = \{q_{\text{unsafe}}\}$

q_{unsafe} represents the state where $c(t)$ is 1, meaning that the delay level of one routing path exceeds its threshold and the other routing path is still recovering. Verifying whether we can keep the connection (at least one routing path not congested) between the source and the destination is reduced to solving for the reachability of q_{unsafe} for the finite state machine. Similar to the discrete second-order control, we compare those approaches as well as the optimal verified set obtained by the optimal approach.

Table 4. Network routing: runtime (in second) with different values of $|Q|$.

$\|Q\|$	BF	MONO	MONO-DUB	LCF	OPT	L-BFS	DL-BFS
678	9.421	1.930	1.229	1.229	0.513	1.196	0.924
1,238	23.362	5.595	3.599	3.599	3.485	4.051	3.378
1,842	36.928	8.630	6.340	6.340	5.345	6.928	6.266
2,452	43.097	11.263	8.314	8.314	8.283	9.284	9.572
3,062	50.155	15.406	11.237	11.237	9.847	13.336	11.835

Table 5. Network routing: runtime (in second) with different values of K.

K	BF	MONO	MONO-DUB	LCF	OPT	L-BFS	DL-BFS
12	0.599	0.260	0.160	0.160	0.115	0.189	0.147
14	2.329	0.858	0.539	0.539	0.493	0.568	0.471
16	9.298	2.855	1.875	1.875	1.829	2.243	1.826
18	23.362	5.595	3.599	3.599	3.485	4.051	3.378
20	129.864	31.260	20.351	20.351	17.040	22.938	19.651

6.2.2 Experiment on $|Q|$. We experimented on how each approach scales with respect to the number of states in the finite-state machine, $|Q|$. To create different numbers of states, we fixed $\tau_1 = 20$ and $\tau_2 = 16$ and experimented with $\gamma = \{20, 30, 40, 50, 60\}$, resulting $|Q|$ from 678 to 3,062. A larger γ makes it more difficult to keep the connection between the source and the destination. K is set to 18. The results are shown in Table 4, and all approaches generate the same satisfaction boundaries. Similar to the previous case study, the proposed approaches outperform the brute-force approach significantly. The monotonic approach with dynamic upper bound of satisfaction boundary and the lowest-cost-first heuristic are generally good in this case study. The dual-layered BFS approach is also good, even using less runtime than the optimal verified set obtained by the optimal approach. It should be noted that an optimal approach defined in Definition 8 only considers approaches which consider weakly-hard constraints one by one and utilize some implications between weakly-hard constraints. Therefore, an approach considering multiple weakly-hard constraints, such as the dual-layered BFS approach, may use less runtime than an optimal approach.

6.2.3 Experiment on K. We experimented on how each approach scales with respect to K. We fixed $\tau_1 = 20$, $\tau_2 = 16$, and $\gamma = 40$. The results are shown in Table 5, where we report the results with $K = 12, 14, 16, 18, 20$. Similar to the previous case study, the proposed approaches outperform the brute-force approach significantly. Among them, the dual-layered BFS approach has the smallest runtimes.

6.3 Lane Changing

6.3.1 Setting. The case study is lane changing with two acceleration controllers on two vehicles driving on two lanes along a road segment with length $x_{\max}$. The following parameters are in meter, second, or their combinations. We denote the position, velocity, and acceleration of the vehicle on the primary lane at t as $x(t)$, $v(t)$, and $a(t)$, and those of the vehicle on the secondary lane as $x'(t)$, $v'(t)$, and $a'(t)$, respectively. Each vehicle receives messages including the position, velocity, and acceleration of the other vehicle. The objective of a controller is to perform lane changing while each vehicle may miss some messages from the other vehicle. A controller is formally defined as $\langle v_{\max}, a_{\min}, a_{\max} \rangle$, where

- $[0, v_{\max}]$ is the physical constraint for the velocity. If the controller attempts to set v to a value larger (smaller) than $v_{\max}$ (0), v is set to the corresponding limit.
- $[a_{\min}, a_{\max}]$ is the acceleration range.

We introduce the following variables:

- $c(t) \in \{0, 1\}$ denotes whether lane changing has happened.
- $s(t) \in \{0, 1\}$ denotes whether the vehicle on the primary lane successfully receives a message from the vehicle on the secondary lane.
- $s'(t) \in \{0, 1\}$ denotes whether the vehicle on the secondary lane successfully receives a message from the vehicle on the primary lane.
- l denotes the length of a vehicle.

The transition functions of the controller on the main lane can be expressed as

$$x(t+1) \leftarrow \min\left(x(t) + v(t) + \frac{1}{2} \cdot a(t), x_{\max}\right), \tag{14}$$

$$v(t+1) \leftarrow \max\left(\min\left(v(t) + a(t), v_{\max}\right), 0\right), \tag{15}$$

$$a(t+1) \leftarrow \begin{cases} 0, & \text{if } s(t) = 0; \\ a_{\min}, & \text{if } s(t) = 1, |x'(t) - x(t)| < 2l, v(t) < v'(t); \\ a_{\max}, & \text{if } s(t) = 1, |x'(t) - x(t)| < 2l, v(t) \geq v'(t); \\ a(t), & \text{otherwise.} \end{cases} \tag{16}$$

The transition functions of the controller on the secondary lane can be expressed as

$$x'(t+1) \leftarrow \min\left(x'(t) + v'(t) + \frac{1}{2} \cdot a'(t), x_{\max}\right), \tag{17}$$

$$v'(t+1) \leftarrow \max\left(\min\left(v'(t) + a'(t), v_{\max}\right), 0\right), \tag{18}$$

$$a'(t+1) \leftarrow \begin{cases} a_{\max}, & \text{if } s'(t) = 0; \\ a_{\min}, & \text{if } s'(t) = 1, |x(t) - x'(t)| < 2l, v(t) \geq v'(t); \\ a_{\max}, & \text{if } s'(t) = 1, |x(t) - x'(t)| < 2l, v(t) < v'(t); \\ a'(t), & \text{otherwise.} \end{cases} \tag{19}$$

We also introduce the following transition function:

$$c(t+1) \leftarrow \begin{cases} 1, & \text{if } c(t) = 1; \\ 1, & \text{if } c(t) = 0, x(t) \neq x_{\max} \text{ or } x'(t) \neq x_{\max}, |x'(t) - x(t)| \geq 2l; \\ 0, & \text{otherwise.} \end{cases} \tag{20}$$

The safety property is $c(t) = 1$, meaning that lane changing has completed, or $x(t) \neq x_{\max}$ or $x'(t) \neq x_{\max}$, meaning that the vehicles have not reached the end of the road segment.

For any controller configuration $\langle v_{\max}, a_{\min}, a_{\max}, v'_{\max}, a'_{\min}, a'_{\max}\rangle$, we can determine a finite state machine $\langle Q, \Sigma, \delta, q_0, F\rangle$, where

- $Q = \{(x, v, a, x', v', a') | x, x' \in [0, x_{\max}], v, v' \in [0, v_{\max}], a, a' \in [a_{\min}, a_{\max}]\}$.
- $\Sigma : \{00, 01, 10, 11\}$, which is the input as $s(t)$ and $s'(t)$.
- δ is defined exactly from the transition functions above.
- $q_0 = (0, 0, 0, 0, 0, 0)$.
- $F = \{q_{\text{unsafe}}\}$

Table 6. Lane changing: runtime (in second) with different values of $|Q|$.

$\|Q\|$	BF	MONO	MONO-DUB	LCF	OPT	L-BFS	DL-BFS
46,835	19.987	2.794	2.365	2.365	1.747	2.584	1.708
72,333	31.002	4.342	3.687	3.687	2.737	3.957	2.817
97,206	40.772	7.880	4.166	4.166	4.077	5.120	3.863
125,152	54.559	10.698	5.453	5.453	5.330	7.240	4.623
155,941	67.018	13.447	7.025	7.025	6.879	8.192	5.980
189,535	81.403	15.820	8.456	8.456	8.294	10.557	7.047

Table 7. Lane changing: runtime (in second) with different values of K.

K	BF	MONO	MONO-DUB	LCF	OPT	L-BFS	DL-BFS
4	2.739	0.763	0.280	0.280	0.280	0.640	0.416
5	6.485	1.978	0.872	0.872	0.709	1.360	0.840
6	15.600	3.298	0.898	0.898	0.721	2.640	1.700
7	34.249	5.514	3.155	3.155	2.978	5.131	3.264
8	81.403	15.820	8.456	8.456	8.294	10.557	7.047

q_{unsafe} represents the state where $x(t) = x'(t) = x_{\max}$ and $c(t) = 0$. Verifying whether the two vehicles can successfully change their lanes is reduced to solving for the reachability of q_{unsafe} for the finite state machine. Similar to the previous case studies, we compare those approaches as well as the optimal verified set obtained by the optimal approach.

6.3.2 Experiment on $|Q|$. We experimented on how each approach scales with respect to the number of states in the finite-state machine, $|Q|$. To create different numbers of states, we fixed $v_{\max} = v'_{\max} = 10$, $a_{\max} = a'_{\max} = 5$, $a_{\min} = a'_{\min} = -5$, and $l = 4$ and experimented with $x_{\max} = \{50, 60, 70, 80, 90, 100\}$, resulting $|Q|$ from 46,835 to 189,535. A larger range $x_{\max}$ of the control value x allows the controller to change the lane more easily. K is set to 8. The results are shown in Table 6, and all approaches generate the same satisfaction boundaries. Similar to the previous case studies, the proposed approaches outperform the brute-force approach significantly. The dual-layered BFS approach is especially good in this case study, even using less runtime than the optimal verified set obtained by the optimal approach in the most cases. Similarly, an optimal approach defined in Definition 8 only considers approaches which consider weakly-hard constraints one by one and utilize some implications between weakly-hard constraints. Therefore, an approach considering multiple weakly-hard constraints, such as the dual-layered BFS approach, may use less runtime than an optimal approach.

6.3.3 Experiment on K. We experimented on how each approach scales with respect to K. We fixed $x_{\max} = 100$, $v_{\max} = v'_{\max} = 10$, $a_{\max} = a'_{\max} = 5$, $a_{\min} = a'_{\min} = -5$, and $l = 4$. The results are shown in Table 7, where we report the results with $K = 4, 5, 6, 7, 8$. Similar to the previous case studies, the proposed approaches outperform the brute-force approach significantly. Among them, the monotonic approach with dynamic upper bound of satisfaction boundary, the lowest-cost-first heuristic, and the dual-layered BFS approach have smaller runtimes.

6.4 Summary

The three case studies demonstrate the applicability and scalability of weakly-hard fault modeling and the proposed approaches. Based on the case studies, the monotonic approach with dynamic upper bound of satisfaction boundary, the lowest-cost-first heuristic, and the dual-layered BFS

approach generally have better efficiency. It should be mentioned that the dual-layered BFS approach is especially for reachability analysis for finite-state machines, so it is not applicable to general properties and general systems. The applicability of them is summarized in Section 1.3 and Table 1.

7 RELATED WORK

Starting from [11], which is the first work that introduced the notion of (m, k) constraint, weakly-hard systems have been studied from various perspectives in the last two decades. Research interests range from real-time systems [2] to network systems [16]. Most of the works focus on the schedulability analysis for periodic tasks under various assumptions such as bi-modal execution and non-preemptiveness [3, 5, 18, 25], or the temporal behavior analysis of overloaded systems [1, 10, 12, 23, 27].

Stable controller synthesis is another important topic in the context of weakly-hard constraints. Based on the extensive studies on the stability under probabilistic deadline misses [22, 24], authors in [4] propose a switched controller to stabilize a weakly-hard system with linear dynamic, while a non-switched controller is discussed in [21].

The most related work is the safety verification for weakly-hard systems, where however, only a few prior works have been devoted to this topic. [8] was the first work that attempts to provide a formal analysis for linear dynamical systems with weakly-hard constraints. In this paper, a weakly-hard system with linear dynamic is modeled as a hybrid automaton and then the reachability of the generated hybrid automaton is verified by the tool SpaceEx [9]. [7] transforms the behavior of a linear weakly-hard system into a program, and then uses program verification techniques, such as abstract interpretation and SMT solvers to analyze the safety. In contrast, the infinite-time safety problem of general nonlinear weakly-hard systems is considered in [15]. By modeling a weakly-hard system as a hybrid automaton, which is similar to that in [8], authors in [15] convert the infinite-time safety problem into a finite one and then apply linear programming to obtain a sufficient condition of the initial state to ensure the safety, which is further improved in [14].

The fundamental difference between the above works, and this paper, is that we focus on discrete systems rather than continuous systems. Since a variety of systems are discrete in practice, we believe the study on specific discrete systems is necessary. Benefiting from this, our technique can generate sound and complete verification results with respect to the weakly-hard constraints for large scale problems.

8 CONCLUSION

In this paper, we used a weakly-hard fault model to constrain the occurrences of faults in system inputs. We developed approaches to verify properties for multiple weakly-hard constraints in an exact and efficient manner. By verifying multiple weakly-hard constraints and storing the verification results as a safety table or the corresponding satisfaction boundary, we defined weakly-hard requirements for the system environment and designed a runtime monitor that guarantees desired properties or notifies the system to switch to a safe mode. Experimental results with discrete second-order control, network routing, and lane changing demonstrated the generality and the efficiency of the proposed approaches. Future directions include properties in LTL under weakly-hard constraints, other models of computation (such as timed automata or hybrid systems to catch the notion of time, although weakly-hard constraints will be used to model the discrete parts of timed automata or hybrid systems due to the nature of weakly-hard constraints which count the numbers of some events) under weakly-hard constraints, and system-specific cost estimation for the lowest-cost-first heuristic.

ACKNOWLEDGEMENTS

This work is supported by the Asian Office of Aerospace Research and Development (AOARD), jointly with the Office of Naval Research Global (ONRG), award FA2386-19-1-4037, the Taiwan Ministry of Education (MOE) grant NTU-111V1901-5, the Taiwan National Science and Technology Council (NSTC) grants NSTC-111-2636-E-002-018 and NSTC-112-2636-E-002-010. It is also supported by the US National Science Foundation (NSF) grants CNS-2038853 and CCF-2144860, and the Office of Naval Research (ONR) grant N00014-19-1-2496.

REFERENCES

[1] L. Ahrendts, S. Quinton, T. Boroske, and R. Ernst. 2018. Verifying weakly-hard real-time properties of traffic streams in switched networks. In *Euromicro Conference on Real-Time Systems*, Vol. 106. 15:1–15:22.

[2] G. Bernat, A. Burns, and A. Liamosi. 2001. Weakly hard real-time systems. *IEEE Trans. Comput.* 50, 4 (2001), 308–321.

[3] G. Bernat and R. Cayssials. 2001. Guaranteed on-line weakly-hard real-time systems. In *IEEE Real-Time Systems Symposium*. IEEE, 22–35.

[4] R. Blind and F. Allgöwer. 2015. Towards networked control systems with guaranteed stability: Using weakly hard real-time constraints to model the loss process. In *IEEE Conference on Decision and Control*. IEEE, IEEE, 7510–7515.

[5] H. Choi, H. Kim, and Q. Zhu. 2019. Job-class-level fixed priority scheduling of weakly-hard real-time systems. In *IEEE Real-Time Technology and Applications Symposium*. IEEE, 241–253.

[6] H. Choi, H. Kim,, and Q. Zhu. 2021. Toward practical weakly hard real-time systems: a job-class-level scheduling approach. *IEEE Internet of Things Journal* 8, 8 (2021), 6692–6708.

[7] P. S. Duggirala and M. Viswanathan. 2015. Analyzing real time linear control systems using software verification. In *IEEE Real-Time Systems Symposium*. IEEE, IEEE, 216–226.

[8] G. Frehse, A. Hamann, S. Quinton, and M. Woehrle. 2014. Formal analysis of timing effects on closed-loop properties of control software. In *IEEE Real-Time Systems Symposium*. IEEE, 53–62.

[9] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler. 2011. SpaceEx: scalable verification of hybrid systems. In *International Conference on Computer-Aided Verification*. Springer, Springer, 379–395.

[10] A. Gujarati, M. Nasri, R. Majumdar, and B. Brandenburg. 2019. From iteration to system failure: characterizing the FITness of periodic weakly-hard systems. In *Euromicro Conference on Real-Time Systems*. 9:1–9:23.

[11] M. Hamdaoui and P. Ramanathan. 1995. A dynamic priority assignment technique for streams with (m, k)-firm deadlines. *IEEE Trans. Comput.* 44, 12 (1995), 1443–1451.

[12] Z. A. H. Hammadeh, R. Ernst, S. Quinton, R. Henia, and L. Rioux. 2017. Bounding deadline misses in weakly-hard real-time systems with task dependencies. In *Design, Automation and Test in Europe Conference*. 584–589.

[13] Z. A. H. Hammadeh, S. Quinton, M. Panunzio, R. Henia, L. Rioux, and R. Ernst. 2017. Budgeting under-specified tasks for weakly-hard real-time systems. In *Euromicro Conference on Real-Time Systems*, Vol. 76. 17:1–17:22.

[14] C. Huang, K.-C. Chang, C.-W. Lin, and Q. Zhu. 2020. SAW: a tool for safety analysis of weakly-hard systems. In *Computer Aided Verification*, Shuvendu K. Lahiri and Chao Wang (Eds.). Springer International Publishing, Cham, 543–555.

[15] C. Huang, W. Li, and Q. Zhu. 2019. Formal verification of weakly-hard systems. In *ACM International Conference on Hybrid Systems: Computation and Control*. ACM, 197–207.

[16] C. Huang, K. Wardega, W. Li, and Q. Zhu. 2019. Exploring weakly-hard paradigm for networked systems. In *Workshop on Design Automation for CPS and IoT*. 51–59.

[17] V. Lesi, I. Jovanov, and M. Pajic. 2017. Network scheduling for secure cyber-physical systems. In *IEEE Real-Time Systems Symposium*. IEEE, 45–55.

[18] J. Li, Y. Song, and F. Simonot-Lion. 2006. Providing real-time applications with graceful degradation of QoS and fault tolerance according to (m, k)-firm model. *IEEE Transactions on Industrial Informatics* 2, 2 (2006), 112–119.

[19] H. Liang, Z. Wang, R. Jiao, and Q. Zhu. 2020. Leveraging weakly-hard constraints for improving system fault tolerance with functional and timing guarantees. In *IEEE/ACM International Conference On Computer Aided Design*. 1–9.

[20] H. Liang, Z. Wang, D. Roy, S. Dey, S. Chakraborty, and Q. Zhu. 2019. Security-driven codesign with weakly-hard constraints for real-time embedded systems. In *IEEE International Conference on Computer Design*. IEEE, 217–226.

[21] S. Linsenmayer and F. Allgower. 2017. Stabilization of networked control systems with weakly hard real-time dropout description. In *IEEE Conference on Decision and Control*. IEEE, 4765–4770.

[22] P. Pazzaglia, C. Mandrioli, M. Maggio, and A. Cervin. 2019. DMAC: deadline-miss-aware control. In *Euromicro Conference on Real-Time Systems*. 1:1–1:24.

[23] S. Quinton and R. Ernst. 2012. Generalized weakly-hard constraints. In *International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*. Springer, 96–110.

[24] L. Schenato. 2009. To zero or to hold control inputs with lossy links? *IEEE Trans. Automat. Control* 54, 5 (2009), 1093–1099.

[25] Y. Sun and M. Di Natale. 2017. Weakly hard schedulability analysis for fixed priority scheduling of periodic real-time tasks. *ACM Transactions on Embedded Computing Systems* 16, 5s (2017), 171:1–171:19.

[26] S.-L. Wu, C.-Y. Bai, K.-C. Chang, Y.-T. Hsieh, C. Huang, C.-W. Lin, E. Kang, and Q. Zhu. 2020. Efficient system verification with multiple weakly-hard constraints for runtime monitoring. In *International Conference on Runtime Verification*, Jyotirmoy Deshmukh and Dejan Ničković (Eds.). Springer, 497–516.

[27] W. Xu, Z. A. H. Hammadeh, A. Kröller, R. Ernst, and S. Quinton. 2015. Improved deadline miss models for real-time systems using typical worst-case analysis. In *Euromicro Conference on Real-Time Systems*. 247–256.

[28] Q. Zhu, W. Li, H. Kim, Y. Xiang, K. Wardega, Z. Wang, Y. Wang, H. Liang, C. Huang, J. Fan, and H. Choi. 2020. Know the unknowns: addressing disturbances and uncertainties in autonomous systems. In *IEEE/ACM International Conference On Computer Aided Design*.