Efficient System Verification with Multiple Weakly-Hard Constraints for Runtime Monitoring

Shih-Lun Wu[1], Ching-Yuan Bai[1], Kai-Chieh Chang[1], Yi-Ting Hsieh[1], Chao Huang[2], Chung-Wei Lin[1](✉), Eunsuk Kang[3], and Qi Zhu[2]

[1] National Taiwan University, Taipei, Taiwan
{b06902080,b05502055,r08922054,b05902031}@ntu.edu.tw, cwlin@csie.ntu.edu.tw

[2] Northwestern University, Evanston, USA
{chao.huang,qzhu}@northwestern.edu

[3] Carnegie Mellon University, Pittsburgh, USA
eunsukk@andrew.cmu.edu

Abstract. 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. Experimental results with a discrete second-order controller demonstrate the efficiency of the proposed approaches.

Keywords: Formal verification · Weakly-hard models

1 Introduction

Weakly-hard models have been studied in a number of works for real-time systems [1–3,5,9,10,12,18,24], mostly from the perspective of scheduling

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) grants NTU-108V0901 and NTU-107V0901, the Taiwan Ministry of Science and Technology (MOST) grants MOST-109-2636-E-002-022 and MOST-108-2636-E-002-011. It is also supported by the National Science Foundation (NSF) awards CCF-1918140, CNS-1834701, CNS-1801546, and the Office of Naval Research (ONR) grant N00014-19-1-2496.
Shih-Lun Wu and Ching-Yuan Bai contributed equally.

J. Deshmukh and D. Ničković (Eds.): RV 2020, LNCS 12399, pp. 497–516, 2020.
https://doi.org/10.1007/978-3-030-60508-7_28

constraints. In this paper, we use a weakly-hard model to constrain the occurrences of faults, and verify properties of discrete systems under such weakly-hard fault model. In particular, we leverage the (m, k) constraint for fault modeling $(0 \leq m \leq k)$, which specifies that there are at most m bad events (faults) among any k consecutive events. Verifying system properties under this fault model has various applications, such as the ones below:

– In a real-time system, a deadline miss can be considered as a bad event (fault). Our approach can help find the maximum number of deadline misses allowed for ensuring 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 approach 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 [16].
– 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 the attackers, our approach can be applied to minimize attacking cost while still causing the system to reach a state violating properties.

More generally speaking, our verification approach under the (m, k) weakly-hard fault model provides two important properties for system engineering:

– If the environment and system design (*e.g.*, via scheduling) ensures 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 really desired.

In this paper, given a labelled transition system S, a property P, and a positive integer K, we aim to develop a runtime monitor to verify whether the environment satisfies a subset of the (m, k) constraints, where $1 \leq m \leq k \leq K$ and the subset is sufficient to enforce P, *i.e.*, if the environment satisfies the subset of the (m, k) constraints, it implies that S guarantees to satisfy P; otherwise, S cannot guarantee to satisfy P, which should lead S to switch to a safe mode. Different from some existing runtime-monitoring approaches that do not have the model of S, in this paper the model of S is given, but the satisfaction of an (m, k) constraint can only be verified during runtime.

The runtime monitor relies on a *safety table* which stores the satisfaction condition of the property under each (m, k) constraint. As there are $\frac{K(K+1)}{2}$ 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 include:

- We derive theorems of logical relationships between weakly-hard constraints. Based on the logical relationships, we reduce a safety table to its *satisfaction boundary* and propose approaches which only need to verify the property at most $2K$ times to compute the satisfaction boundary.
- Based on the computed satisfaction boundary, we define weakly-hard requirements for the system environment and design a lightweight runtime monitor monitoring the satisfaction of the weakly-hard requirements.
- We consider a special case of reachability of finite-state machines. We propose a mask-compressing approach which can be plugged into (called by) the proposed approaches above. We further 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.
- Experiment results with a discrete second-order controller demonstrate the efficiency of the proposed approaches.

The paper is organized as follows. Section 2 provides the problem formulation, and Sect. 3 overviews the proposed approaches. Section 4 describes how we solve the problem for general properties and systems and design a runtime monitor. 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 Sect. 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 $Q_0 \in Q$ is the set of initial states. Without loss of generality, a subset of 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.

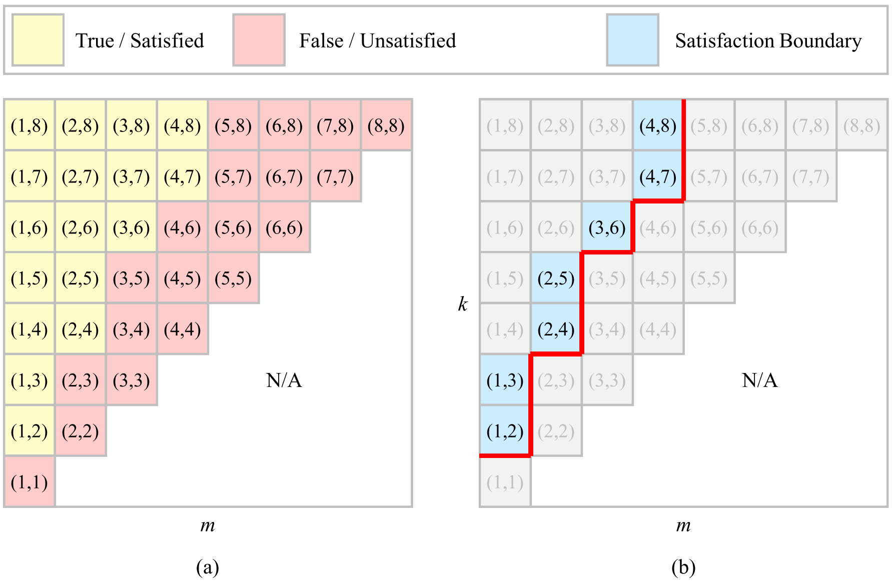
True / Satisfied
False / Unsatisfied
Satisfaction Boundary
(1,8) (2,8) (3,8) (4,8) (5,8) (6,8) (7,8) (8,8)
(1,7) (2,7) (3,7) (4,7) (5,7) (6,7) (7,7)
(1,6) (2,6) (3,6) (4,6) (5,6) (6,6)
(1,5) (2,5) (3,5) (4,5) (5,5)
(1,4) (2,4) (3,4) (4,4)
(1,3) (2,3) (3,3)
(1,2) (2,2)
(1,1)
N/A
k
m
(a)
(b)

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

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 in this paper is to develop a runtime monitor to verify whether the environment satisfies a subset of $C(K)$, where the subset is sufficient to enforce P, *i.e.*, if the environment satisfies the subset of $C(K)$, it implies that S guarantees to satisfy P; otherwise, 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 fit it).

The runtime monitor relies on a *safety table*, which stores the satisfaction condition of P under each $W(m, k)$ in $C(K)$. A safety table is defined as follows.

Table 1. The proposed approaches, where the monotonic approach (Algorithm 1), the monotonic approach with dynamic upper bound of satisfaction boundary (Algorithm 2), and the lowest-cast-first heuristic (Algorithm 3) decide the order of evaluating the weakly-hard constraints and need to call a verification approach (not covered for general properties and general systems in this paper) for a single (m, k) constraint to complete the verification for multiple (m, k) constraints.

Property & System	Single (m, k) Constraint	Multiple (m, k) Constraints
Reachability & Finite-State Machine	Mask-Compressing (Sect. 5.2)	Layered BFS (Sect. 5.3)
General Property & General System	Not Covered	Algorithms 1, 2, and 3 (Sects. 4.3, 4.5, and 4.6)

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

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

For $m > k$, $T[m, k]$ is not applicable as the corresponding weakly-hard fault model is undefined. Note the a safety table is computed off-line in 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 safety table is shown in Fig. 1(a).

3 Overview of Proposed Approaches

We list the proposed approaches in this paper in Table 1. There will be five approaches: the monotonic approach (Algorithm 1) in Sects. 4.3, the monotonic approach with dynamic upper bound of satisfaction boundary (Algorithm 2) in Sects. 4.5, the lowest-cast-first heuristic (Algorithm 3) in Sects. 4.6, the mask-compressing approach in Sects. 5.2, and the layered BFS approach in Sects. 5.3.

The first three approaches are for general properties, general systems, and multiple weakly-hard constraints. They decide the order 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 two 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 is for multiple weakly-hard constraints.

4 General Approaches and Runtime Monitor Design

In this section, we first define the strength of weakly-hard constraints (Sect. 4.1). We then derive the fundamental theorems of logical relationships between weakly-hard constraints (Sect. 4.2) and propose an algorithm to compute the safety table and its corresponding satisfaction boundary based on these theorems (Sect. 4.3). We further derive advanced theorems of logical relationships between weakly-hard constraints (Sect. 4.4) and propose an improved algorithm (Sect. 4.5) and a lowest-cost-first heuristic (Sect. 4.6) taking all properties into account. Based on the computed safety table and the satisfaction boundary, we can design a runtime monitor (Sect. 4.7).

4.1 Strength of Weakly-Hard Constraint

Definition 4. ***Strength of Weakly-Hard Constraint***. *Given two 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.

4.2 Fundamental Theorems

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 $\sigma \models W(m',k)$.

Corollary 1. *For any $m, m', k \in \mathbb{Z}^+, m < m' \leq k$, if a property P is not satisfied under $W(m,k)$, then P is not satisfied 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 $\sigma \models W(m,k')$.

Corollary 2. *For any $m, k, k' \in \mathbb{Z}^+, m \leq k' < k$, if a property P is not satisfied under $W(m,k)$, then P is not satisfied under $W(m,k')$; if a property P is satisfied under $W(m,k')$, then P is satisfied under $W(m,k)$.*

Algorithm 1. Monotonic Approach

```
procedure GET_SATISFACTION_BOUNDARY($S, P, K$)
    $B \leftarrow [\,]$
    $m \leftarrow 0$
    for $k \leftarrow 1$ to $K$ do                    ▷ Get satisfaction boundary for each $k$
        while $m < k$ do
            if $S \not\models P$ under $W(m+1, k)$ then
                break
            end if
            $m \leftarrow m + 1$
        end while
        $B[k] \leftarrow m$
    end for
    return $B$
end procedure
```

By Corollary 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 Fig. 1(a) is shown in Fig. 1(b). The reduction is crucial because we only need to store the satisfaction boundary rather than the whole safety table for the runtime monitor.

4.3 Monotonic Approach

Corollaries 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 Sect. 5.

We propose 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 not satisfied and obtains $B(k)$ (Lines 5–11). By Corollary 1, since P is not satisfied under $W(B(k)+1, k)$, P is not satisfied 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 Fig. 2(a), if P is not satisfied under $W(3, 4)$, then P is not satisfied 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 Corollary 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 Fig. 2(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

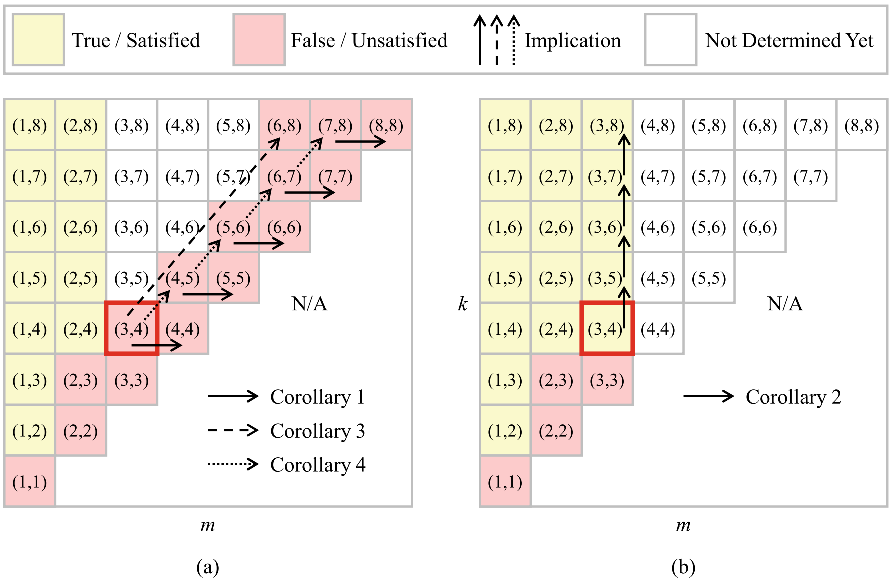

Fig. 2. An illustration of Algorithms 1 (which applies Corollaries 1 and 2 only) and 2 (which applies Corollaries 1, 2, 3, and 4). To have a clear comparison, we focus on the implications of $W(3,4)$ only. (a) If P is not satisfied under $W(3,4)$, then P is not satisfied under $W(4,4)$. Algorithm 2 further implies that P is not satisfied 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 $k \geq 5$.

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 $O(K^2 \cdot X)$.

4.4 Advanced Theorems

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

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 $\sigma \models W(xm, xk)$.

Corollary 3. *For any $m, k, x \in \mathbb{Z}^+, m < k, x \geq 2$, if a property P is not satisfied under $W(m,k)$, then P is not satisfied under $W(xm, xk)$; if a property P is satisfied under $W(xm, xk)$, then P is satisfied under $W(m,k)$.*

Algorithm 2. Monotonic Approach with Dynamic Upper Bound of Satisfaction Boundary

```
procedure GET_SATISFACTION_BOUNDARY($S, P, K$)
    $B \leftarrow [\,]$
    $m \leftarrow 0$
    for $k \leftarrow 1$ to $K$ do                              ▷ Initialize satisfaction boundary
        $B[k] = k$
    end for
    for $k \leftarrow 1$ to $K$ do                              ▷ Get satisfaction boundary for each $k$
        while $m < B[k]$ do
            if $S \not\models P$ under $W(m+1, k)$ then
                $x \leftarrow 2$
                while $x \cdot k \leq K$ do                           ▷ Corollary 3
                    $B[xk] \leftarrow \min(B[xk], x \cdot (m+1) - 1)$
                    $x \leftarrow x + 1$
                end while
                $x \leftarrow 1$
                while $k + x \leq K$ do                           ▷ Corollary 4
                    $B[k+x] \leftarrow \min(B[k+x], (m+1) + x - 1)$
                    $x \leftarrow x + 1$
                end while
                break
            end if
            $m \leftarrow m + 1$
        end while
        $B[k] \leftarrow \min(B[k], m)$
    end for
    return $B$
end procedure
```

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

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 $\sigma \models W(m+x, k+x)$.

Corollary 4. *For any $m, k, x \in \mathbb{Z}^+, m < k$, if a property P is not satisfied under $W(m, k)$, then P is not satisfied under $W(m + x, k + x)$; if a property P is satisfied under $W(m + x, k + x)$, then P is satisfied under $W(m, k)$.*

4.5 Monotonic Approach with Dynamic Upper Bound of Satisfaction Boundary

Corollaries 3 and 4 imply the satisfaction of a property P beyond the same m or k. Integrating with the previously proposed monotonic approach which

Algorithm 3. Lowest-Cost-First Heuristic

```
1: procedure GET_SAFETY_TABLE($S, P, K$)
2:     $T \leftarrow \{$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 $S \models P$ under $W(m, k)$ then
6:             $T[m, k] \leftarrow$ True
7:         else
8:             $T[m, k] \leftarrow$ False
9:         end if
10:        Recursively update $T$ by Corollaries 1, 2, 3, and 4
11:    end while
12:    return $T$
13: end procedure
```

increases m and k, we exploit the corollaries and propose Algorithm 2 to compute the satisfaction boundary $B(k)$ for each $k \leq K$. The main difference between Algorithm 1 and Algorithm 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 not satisfied under a constraint.

Specifically, suppose the algorithm is in the process of computing $B(k)$, and P is not satisfied under $W(m+1, k)$ (Line 9). By Corollary 3, P is not satisfied 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 Corollary 4, P is not satisfied 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 Fig. 2(a), if P is not satisfied under $W(3, 4)$, then P is not satisfied under $W(4, 4)$, $W(6, 8)$, and $W(m, k)$ where $k \geq 5$ and $m \geq k-1$, which 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 Fig. 2(b).

4.6 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 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 more constraints or higher-cost constraints is preferred. We propose 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.

Algorithm 4. Runtime Monitoring

```
1: procedure RUNTIME_MONITORING($K, B[]$)
2:     for $k \leftarrow 1$ to $K$ do
3:         $I[k] \leftarrow 0$                      ▷ Store the last $k$-th input
4:         $N_1[k] \leftarrow 0$                    ▷ Store the number of 1's among the last $k$ inputs
5:     end for
6:     $i \leftarrow 0$
7:     while 1 do                                   ▷ During runtime
8:         $x$ = Get_Input()
9:         for $k \leftarrow 1$ to $K$ do
10:            $N_1[k] \leftarrow N_1[k] + x - I[(i-k)\%K]$
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:        $I[i] \leftarrow x$
16:        $i \leftarrow (i+1)\%K$
17:    end while
18: end procedure
```

4.7 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 check 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.

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-compression approach to verify reachability under a single weakly-hard constraint. The mask-compression 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 Sect. 4. Then, we propose a layered

BFS approach which computes the safety table in a more efficient way—the layered BFS approach computes the safety table with the same computational complexity as evaluating a single (m, K) constraint.

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, $P_r : \delta \rightarrow (0, 1]$ is the transition probability satisfying

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

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-compression 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 Sect. 4. 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 Count1() counts the number of 1's in a mask.

Note that Eq. (3) is for the transition in S, Eq. (4) is for the 1-bit "shift" of the mask, and Eq. (5) is for the number of 1's bounded by the weakly-hard fault model. After constructing the graph, we can apply the depth-first search from $v_{q_0,0}$, and P is not satisfied if and only if we can reach a vertex $v_{q,mask}$ where $q \in F$.

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 $W(m, k)$. When plugging the masking-compressing approach into the approaches in Sect. 4, the complexities are as follows:

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

All of them are bounded by

$$O\left(\sum_{k=1}^{K} k \cdot N \cdot 2^k\right) = O\left((K-1) \cdot N \cdot 2^{K+1} + N\right) = O\left(K \cdot N \cdot 2^K\right). \quad (6)$$

5.3 Layered BFS Approach

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 5. *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. It is straightforward by Eq. (5).

Theorem 5 implies that evaluating $W(m, k)$ leads to the results for all $W(m', k)$, where $0 \leq m' \leq m$. Thus, only the graph for $W(k, k)$ needs to be traversed. The problem boils down to finding the correct order to perform graph traversal such that all verification results can be collected. Formally, we let E_m and V_m denote the set of edges and vertices of the graph for $W(m, k)$. At the m-th iteration (as a layer), we perform a 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 $G_{m-1} \subseteq G_m$.

Since we aim to expand the smallest graph for $W(1, k)$ incrementally up to $W(k, k)$ in a bottom-up manner, we iteratively allow parts of the graph for $W(k, k)$ to be "visitable" and perform a BFS on visitable vertices. Initially, $E_0 = \emptyset$ and $V_0 = \{v_{q_0,0^k}\}$. At the beginning of the m-th iteration, the layered BFS approach marks each vertex $v_{q,mask}$, where $mask$ satisfies $W(m, k)$, to be visitable. Then, in the same iteration, the layered BFS approach performs a BFS on visitable vertices to find reachable vertices and mark them to be "reachable".

If an unsafe state is reached at the m-th iteration, P is only guaranteed to be satisfied under $W(m', k)$, where $m' < m$.

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

$$O\left(\sum_{k=1}^{K} N \cdot 2^k\right) = O\left(N \cdot 2^{K+1} - N \cdot 2\right) = O(N \cdot 2^K). \tag{7}$$

This shows that the layered BFS approach computes the satisfaction boundary with the same complexity as verifying a single (m, K). Compared with Algorithms 1, 2, and 3 with the complexity $O(K \cdot N \cdot 2^K)$ in Eq. (6), the layered BFS approach is asymptotically K times faster, demonstrating that white or grey box system models allow more efficient verification.

6 Experiment Results

6.1 Setting

The case study is a discrete second-order controller under perturbation attacks. 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. The controller 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

$$x_{\text{dec}}(t) = \dot{x}(t) \cdot t_{\text{dec}}(t) - \frac{1}{2} \cdot \text{sign}(\dot{x}(t)) \cdot \ddot{x}_C \cdot t_{\text{dec}}(t)^2, \tag{8}$$

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

$$x(t+1) \leftarrow x(t) + \dot{x}(t) + p_{\text{atk}}(t), \tag{9}$$

$$\dot{x}(t+1) \leftarrow \max\Bigg(\min\Big(\dot{x}(t)+\ddot{x}(t), \dot{x}_{\max}\Big), \dot{x}_{\min}\Bigg), \tag{10}$$

$$\ddot{x}(t+1) \leftarrow -\mathrm{sign}\Big(x(t)+p_{\mathrm{atk}}(t)\Big)\cdot \mathrm{sign}\Big(|x(t)+p_{\mathrm{atk}}(t)| - |x_{\mathrm{dec}}(t)|\Big)\cdot \ddot{x}_C, \tag{11}$$

where p_{atk} denotes the perturbation attack. Equation (9) is for the transition of x, where the control value is affected by both the first-order derivative and the perturbation attack. Equation (10) 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 (11) 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.

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

- $Q = \{(x, \dot{x}, \ddot{x}) | x, \dot{x} \in \mathbb{Z}, x \in [x_{\min}, x_{\max}], \dot{x} \in [\dot{x}_{\min}, \dot{x}_{\max}], \ddot{x} \in \{-\ddot{x}_C, 0, \ddot{x}_{\mathrm{C}}\}\} \cup \{q_{\mathrm{unsafe}}\}$.
- $\Sigma = S_{\mathrm{atk}} \cup \{0\}$.
- δ is defined exactly from the transition functions above.
- $P_r((x, \dot{x}, \ddot{x}), p_{\mathrm{atk}}, (x', \dot{x}', \ddot{x}')) = \frac{1}{|S_{\mathrm{atk}}|}$.
- $q_0 = (0, 0, 0)$.
- $F = \{q_{\mathrm{unsafe}}\}$.

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.

We implemented a brute-force approach which evaluates all constraints in $C(K)$ one by one, the monotonic approach (Algorithm 1), the monotonic approach with dynamic upper bound of satisfaction boundary (Algorithm 2), the lowest-cost-first heuristic (Algorithm 3) which defines the estimated cost for evaluating $W(m, k)$ as $\sum_{i=0}^{m}\binom{k-1}{i}$, and the layered BFS approach. Except the 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 were implemented in C++ and run in the environment with 2.4GHz Quad-Core Intel Core i5 CPU and 16GB LPDDR3 RAM. Any reported runtime is the average of 5 runs.

6.2 Results

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 $\dot{x}_{\min} = -4$, $\dot{x}_{\max} = 4$, $\ddot{x}_{\mathrm{C}} = 2$, and $S_{\mathrm{atk}} = \{5\}$ and experimented with $(x_{\min}, x_{\max}) = \{\pm 30, \pm 40, \pm 50, \pm 60, \pm 70, \pm 80, \pm 90, \pm 100\}$, resulting $|Q|$ from 931 to 3,031. 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.

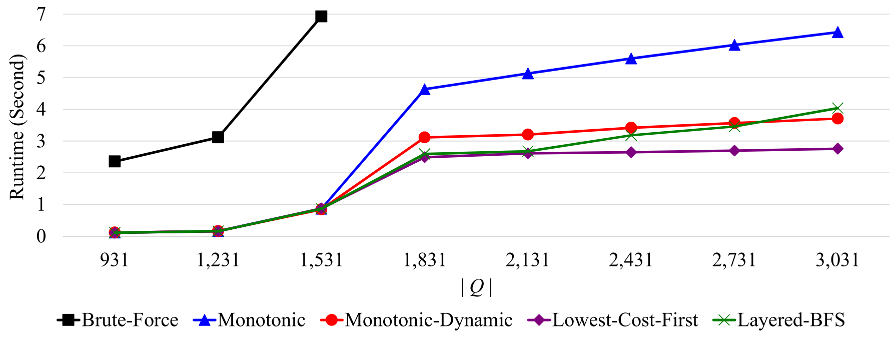

Fig. 3. The runtime over the number of states, $|Q|$ (the out-of-range runtimes of the brute-force approach are 26.972, 29.578, 31.760, 33.975, and 36.047 seconds in an increasing-$|Q|$ order).

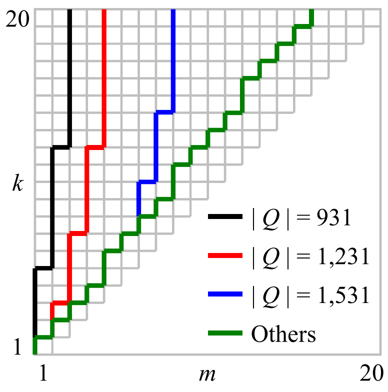

Fig. 4. The computed satisfaction boundaries.

The results are shown in Fig. 3, and the corresponding satisfaction boundaries are illustrated in Fig. 4, 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 Corollaries 1 and 2. For larger number of states, the runtime differences are even larger, and only the monotonic approach can complete the system verification within reasonable time. 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 Corollaries 3 and 4, and the lowest-cost-first heuristic performs faster than the monotonic-dynamic approach when the number of states is larger. The layered BFS approach runs faster than the monotonic approach,

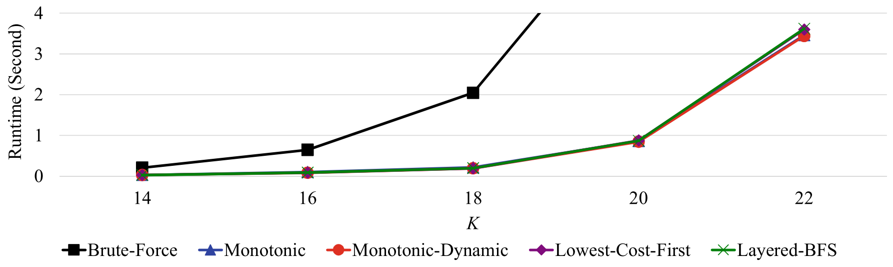

Fig. 5. The runtime over K (the out-of-range runtimes of the brute-force approach are 6.924 and 28.704 seconds in an increasing-K order).

and it has comparable runtime as the monotonic-dynamic approach and the lowest-cost-first heuristic.

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 Fig. 5, 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 is 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.

7 Related Work

Starting from [10], 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 [15]. Most of the works focus on the schedulability analysis for periodic tasks under various assumptions such as bi-modal execution and non-preemptiveness [3,5,17,23], or the temporal behavior analysis of overloaded systems [1,9,11,21,24].

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 [20,22], authors in [4] propose a switched controller to stabilize a weakly-hard system with linear dynamic, while a non-switched controller is discussed in [19].

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. [7] 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 [8]. [6] 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 [14]. By modeling a weakly-hard system as a hybrid automaton, which is similar as that in [8], authors in [14] 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 [13].

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 is able to generate sound and complete verification result 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. Experiments with a discrete second-order controller demonstrated the efficiency of the proposed approaches. Future directions include properties in Linear Temporal Logic under weakly-hard constraints, other models of computation under weakly-hard constraints, and system-specific cost estimation for the lowest-cost-first heuristic.

References

1. Ahrendts, L., Quinton, S., Boroske, T., Ernst, R.: Verifying weakly-hard real-time properties of traffic streams in switched networks. In: Euromicro Conference on Real-Time Systems, vol. 106, pp. 15:1–15:22 (2018)
2. Bernat, G., Burns, A., Liamosi, A.: Weakly hard real-time systems. IEEE Trans. Comput. **50**(4), 308–321 (2001)
3. Bernat, G., Cayssials, R.: Guaranteed on-line weakly-hard real-time systems. In: IEEE Real-Time Systems Symposium, pp. 22–35 (2001)

4. Blind, R., Allgöwer, F.: 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, pp. 7510–7515. IEEE (2015)
5. Choi, H., Kim, H., Zhu, Q.: Job-class-level fixed priority scheduling of weakly-hard real-time systems. In: IEEE Real-Time Technology and Applications Symposium, pp. 241–253 (2019)
6. Duggirala, P.S., Viswanathan, M.: Analyzing real time linear control systems using software verification. In: IEEE Real-Time Systems Symposium, pp. 216–226. IEEE (2015)
7. Frehse, G., Hamann, A., Quinton, S., Woehrle, M.: Formal analysis of timing effects on closed-loop properties of control software. In: IEEE Real-Time Systems Symposium, pp. 53–62 (2014)
8. Frehse, G., et al.: SpaceEx: scalable verification of hybrid systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 379–395. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22110-1_30
9. Gujarati, A., Nasri, M., Majumdar, R., Brandenburg, B.B.: From iteration to system failure: characterizing the fitness of periodic weakly-hard systems. In: Euromicro Conference on Real-Time Systems, pp. 9:1–9:23 (2019)
10. Hamdaoui, M., Ramanathan, P.: A dynamic priority assignment technique for streams with (m, k)-firm deadlines. IEEE Trans. Comput. **44**(12), 1443–1451 (1995)
11. Hammadeh, Z.A.H., Ernst, R., Quinton, S., Henia, R., Rioux, L.: Bounding deadline misses in weakly-hard real-time systems with task dependencies. In: Design, Automation and Test in Europe Conference, pp. 584–589 (2017)
12. Hammadeh, Z.A.H., Quinton, S., Panunzio, M., Henia, R., Rioux, L., Ernst, R.: Budgeting under-specified tasks for weakly-hard real-time systems. In: Euromicro Conference on Real-Time Systems, vol. 76, pp. 17:1–17:22 (2017)
13. Huang, C., Chang, K.-C., Lin, C.-W., Zhu, Q.: SAW: a tool for safety analysis of weakly-hard systems. In: Lahiri, S.K., Wang, C. (eds.) CAV 2020. LNCS, vol. 12224, pp. 543–555. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-53288-8_26
14. Huang, C., Li, W., Zhu, Q.: Formal verification of weakly-hard systems. In: ACM International Conference on Hybrid Systems: Computation and Control, pp. 197–207 (2019)
15. Huang, C., Wardega, K., Li, W., Zhu, Q.: Exploring weakly-hard paradigm for networked systems. In: Workshop on Design Automation for CPS and IoT, pp. 51–59 (2019)
16. Lesi, V., Jovanov, I., Pajic, M.: Network scheduling for secure cyber-physical systems. In: IEEE Real-Time Systems Symposium, pp. 45–55 (2017)
17. Li, J., Song, Y., Simonot-Lion, F.: Providing real-time applications with graceful degradation of QoS and fault tolerance according to (m, k)-firm model. IEEE Trans. Ind. Inf. **2**(2), 112–119 (2006)
18. Liang, H., Wang, Z., Roy, D., Dey, S., Chakraborty, S., Zhu, Q.: Security-driven codesign with weakly-hard constraints for real-time embedded systems. In: 2019 IEEE 37th International Conference on Computer Design (ICCD), pp. 217–226 (2019)
19. Linsenmayer, S., Allgower, F.: Stabilization of networked control systems with weakly hard real-time dropout description. In: IEEE Conference on Decision and Control, pp. 4765–4770 (2017)
20. Pazzaglia, P., Mandrioli, C., Maggio, M., Cervin, A.: DMAC: deadline-miss-aware control. In: Euromicro Conference on Real-Time Systems, pp. 1:1–1:24 (2019)

21. Quinton, S., Ernst, R.: Generalized weakly-hard constraints. In: Margaria, T., Steffen, B. (eds.) ISoLA 2012. LNCS, vol. 7610, pp. 96–110. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34032-1_13
22. Schenato, L.: To zero or to hold control inputs with lossy links? IEEE Trans. Autom. Control **54**(5), 1093–1099 (2009)
23. Sun, Y., Natale, M.D.: Weakly hard schedulability analysis for fixed priority scheduling of periodic real-time tasks. ACM Trans. Embed. Comput. Syst. **16**(5s), 171:1–171:19 (2017)
24. Xu, W., Hammadeh, Z.A.H., Kröller, A., Ernst, R., Quinton, S.: Improved deadline miss models for real-time systems using typical worst-case analysis. In: Euromicro Conference on Real-Time Systems, pp. 247–256 (2015)