國立臺灣大學電機資訊學院資訊工程學研究所

碩士論文

Department of Computer Science and Information Engineering
College of Electrical Engineering and Computer Science
National Taiwan University

Master Thesis

弱硬式網宇實體系統之驗證與排程性分析

Verification and Schedulability Analysis for Cyber-Physical Systems with Multiple Weakly-Hard Constraints

謝議霆

Yi-Ting Hsieh

指導教授：林忠緯 博士

Advisor: Chung-Wei Lin, Ph.D.

中華民國 111 年 6 月

June 2022

doi:10.6342/NTU202200995

Acknowledgements

I would like to express my gratitude to my advisor, Professor Chung-Wei Lin. He gave me a lot of support for the research, and I learned a lot from his tremendous research experience. He also provided opportunities for me to enhance my presentation skills. Moreover, he also cares about the life planning and busyness of all the members of CPS lab. Without his guidance, I cannot complete my education and research.

I would also like to thank Professor Eunsukk and Tzu-Tao Chang. Though the COVID-19 issue in Taiwan, Tzu-Tao had a great contribution to my research and positively discussed with me virtually. Eunsukk gave a lot of advice and method from the innovative direction which also help a lot. I really appreciate their support and am really delighted to work with them.

Yi-Ting Hsieh

National Taiwan University
June 2022

doi:10.6342/NTU202200995

弱硬式網宇實體系統之驗證與排程性分析

研究生：謝議霆　　指導教授：林忠緯 博士

國立臺灣大學資訊工程學研究所

摘要

一個弱硬式錯誤模型可以用(m, k)來描述，代表在 k 個連續的事件中，最多只有 m 個錯誤的事件。這篇論文我們用弱硬式錯誤模型來約束系統的一個輸入，我們提出的方法可以快速的在所有可能的(m, k)下驗證系統的性質，在驗證完所有的(m, k)後，我們可以知道系統需要被約束在甚麼條件下才能保有該性質，並且我們可以設計一個即時監控來偵測目前錯誤的數量，如果偵測到的系統輸入符合弱硬式條件的需求，我們可以保證系統的性質；否則，即時監控可以讓系統進入到一個安全的模式。這對於需要在有限資源和存在錯誤的情況下提供保證的網宇實體系統尤其重要，離散二階控制、網絡路由、車輛跟隨和車道變換的實驗結果證明了所提出方法的普遍性和效率。此外，考慮到多個系統共享一個處理器，需要一個排程器，由於在排程過程中應保證系統的特性，系統設計人員可以將驗證結果中的弱硬約束分配給系統。調度器必須確保資源分配能夠滿足約束，我們還提出了多個弱硬條件下的排

doi:10.6342/NTU202200995

程性分析方法，並將我們的方法與其他排程性分析方法進行比較以證明其通用性。驗證可以為系統設計提供策略，排程性分析可以幫助系統設計人員分析策略是否可行。

關鍵詞：正規式驗證、即時監控、弱硬式模型、排程性分析

doi:10.6342/NTU202200995

VERIFICATION AND SCHEDULABILITY ANALYSIS FOR CYBER-PHYSICAL SYSTEMS WITH MULTIPLE WEAKLY-HARD CONSTRAINTS

Student: Yi-Ting HSIEH Advisor: Dr. Chung-Wei Lin

Department of Computer Science and Information Engineering
National Taiwan University

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 thesis, 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. Moreover, considering multiple systems sharing a processor, a scheduler

doi:10.6342/NTU202200995

is needed. Since properties of systems should be guaranteed during the scheduling process, system designers can assign weakly-hard constraints from the verification results to systems. The scheduler has to make sure the resources distribution can satisfy the constraints. We also propose the schedulability analysis method under multiple weakly-hard constraints and compare our method with other schedulability analysis methods to demonstrate the generality. The verification can give the strategy to system design ,and the schedulability analysis can help system designers analyze whether the strategy is feasible or not.

Keywords: Formal verification, runtime monitoring, weakly-hard models, schedulability analysis

doi:10.6342/NTU202200995

Table of Contents

doi:10.6342/NTU202200995

doi:10.6342/NTU202200995

doi:10.6342/NTU202200995

List of Tables

doi:10.6342/NTU202200995

List of Figures

doi:10.6342/NTU202200995

Chapter 1

Introduction

1.1 Motivations

Cyber-physical systems often tolerate some faults and can still maintain system properties. In this thesis, we constrain the fault occurrence of systems with weakly-hard constraints. Weakly-hard models have different definitions and applications in different works [1–3, 5, 9, 10, 12, 18, 24]. A weakly-hard constraint in this thesis is formatted as (m, k), where among any k consecutive events, there are at most m bad events (faults). We verify properties of systems under the weakly-hard constraint and bring benefits to systems. The tolerance of the bad event can reduce the computation load and the resource can be scheduled to other critical systems. With the verification result, we can optimize resource usage to guarantee system properties.

The verification result under the weakly-hard constraint brings benefits to system designers of cyber-physical systems. If systems can ensure that the fault occurrences satisfy the weakly-hard constraint which guarantees properties, system properties will be guaranteed. If not, system designers should apply a runtime monitor to track whether systems satisfy the weakly-hard constraint. When systems violate the constraint, the runtime monitor can change systems into safe mode and inform the engineers.

doi:10.6342/NTU202200995

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 Related Works

Hamdaoui et al. [10] first introduced the notion of weakly-hard constraints and gave the schedulability analysis to such systems with dynamic priority assignment. Researchers applied the weakly-hard model to different systems, such as real-time systems [2] and network systems [16]. In this thesis, we also applied our approaches to a use case related to network systems.

In this thesis, we consider safety properties verification for weakly-hard cyber-physical systems. [7] proposed a formal analysis for real-time systems by representing them as a network of hybrid automata, and verified by SpaceEx [8]. [6] further reduces the verification problem into software verification. On the other hand, [15] studied the verification problem of weakly-hard systems with nonlinear dynamics and proposed a technique to convert infinite-time safety problems into a finite one. [14] further improved the converting of the infinite-time safety problem with graph theory. In recent works, [23] proposed different approaches to verify the discrete

doi:10.6342/NTU202200995

systems under multiple weakly-hard constraints in an efficient way.

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

[3, 5, 17, 22] focused on the priority assignment algorithm of the scheduling problem and the schedulability analysis under weakly-hard constraints. [22] used fixed priority method while [5, 17] applied dynamic priority assignment method. To bound the temporal behavior of overloaded systems, [1, 9, 11, 13, 19–21, 24], have studied the problem under weakly-hard constraints.

1.3 Contributions

In this thesis, 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. Unlike some existing runtime-monitoring approaches (without an explicit model of S), this thesis assumes that 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 property P under each (m, k) constraint. As there are $\frac{K(K+1)}{2}$ constraints

doi:10.6342/NTU202200995

in the safety table, a straightforward approach evaluating each (m, k) constraint one by one needs to verify a 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 second part of this thesis considers a scheduling problem under weakly-hard constraints. The first part gives the verification result to systems. However, if the consecutive events of multiple systems are the executive jobs and share the same processor, a scheduler will be needed to handle them. Since each system can tolerate some jobs missed (faults), the purpose of the scheduler is to assign the jobs to the processor and maintain the satisfaction of the weakly-hard constraints which can ensure properties of each system.

Considering the scheduling process, the consecutive events of a system in this thesis will be formatted as a task τ_i with period P_i and execution time E_i. The scheduling process is preemptive. That is, P_i can also be the deadline of a job since at each timestamp, the scheduler might assign the different job to the processor. If a job can not be completed before P_i, it will be considered missing the deadline, which is also a fault to a system.

With the safety table, system designers can determine a weakly-hard constraint for systems which will give the least overhead to the scheduling process. However, it is also important that the scheduler can appropriately schedule the taskset. In this thesis, we emphasize the schedulability analysis, which will analyze whether the taskset is schedulable given the taskset, weakly-hard constraints, and the scheduler (priority assignment method). If the taskset is schedulable, we could ensure properties of all systems.

Instead of considering a single weakly-hard constraint for one task, in this

doi:10.6342/NTU202200995

thesis, we consider multiple weakly-hard constraints for one task, which is a more generalized scenario for the scheduling problem. A task with multiple weakly-hard constraints denotes that at each timestamp, the task only needs to satisfy any one of the constraints. Considering multiple constraints can give benefits to system designers. For example, if $(m_1, k_1), (m_2, k_2)$ both ensure system properties, and $(m_1, k_1), (m_2, k_2)$ are not schedulable in single-constraints scenario, it is possible that considering both $(m_1, k_1), (m_2, k_2)$ in the scheduling process will make it schedulable. As the result, considering multiple constraints can bring flexibility to system designers.

The main contributions of this thesis include:

- Based on the existing theorems that state various logical relationships between weakly-hard constraints, we 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. Based on the existing layered Breadth-First Search (BFS) approach, we propose a more efficient dual-layered Breadth-First Search (BFS) approach which computes

doi:10.6342/NTU202200995

Table 1.1: 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 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.

- We propose the schedulability analysis method for multiple weakly-hard constraints given a general priority assignment method.
- Experimental results with discrete second-order control, network routing, and lane changing demonstrate the generality and the efficiency of the proposed approaches.
- Comparing our schedulability analysis method with another method to demonstrate the generality and the similar worst-case analysis.

We overview the existing and proposed approaches in this thesis in Table 1.1. 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, 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

doi:10.6342/NTU202200995

a property P under a single weakly-hard constraint — this thesis 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 is for multiple weakly-hard constraints.

Note that the monotonic approach (Algorithm 1) in Section 3.2, the mask-compressing approach in Section 5.2, and the layered BFS approach in Section 5.3 are the existing approaches refer to [23].

1.4 Organization

The thesis is organized as follows. Chapter 2 provides the problem formulation. Chapter 3 describes how we solve the problem for general properties and systems and introduces a runtime monitor. Chapter 4 discusses optimal approaches. Chapter 5 considers the special case of reachability for finite-state machines. Chapter 6 describes our schedulability analysis method. Chapter 7 presents the experimental results. Chapter 8 concludes the thesis.

doi:10.6342/NTU202200995

Chapter 2

System Model and Problem Formulation

2.1 Weakly-Hard Verification

In this thesis, 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 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.

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 $C(K) \coloneqq \{W(m, k) \mid 1 \leq m \leq k \leq K\}$.*

Given a system S, a property P, and a positive integer K, the goal in this thesis is to develop a runtime monitor to verify whether the environment satisfies a

doi:10.6342/NTU202200995

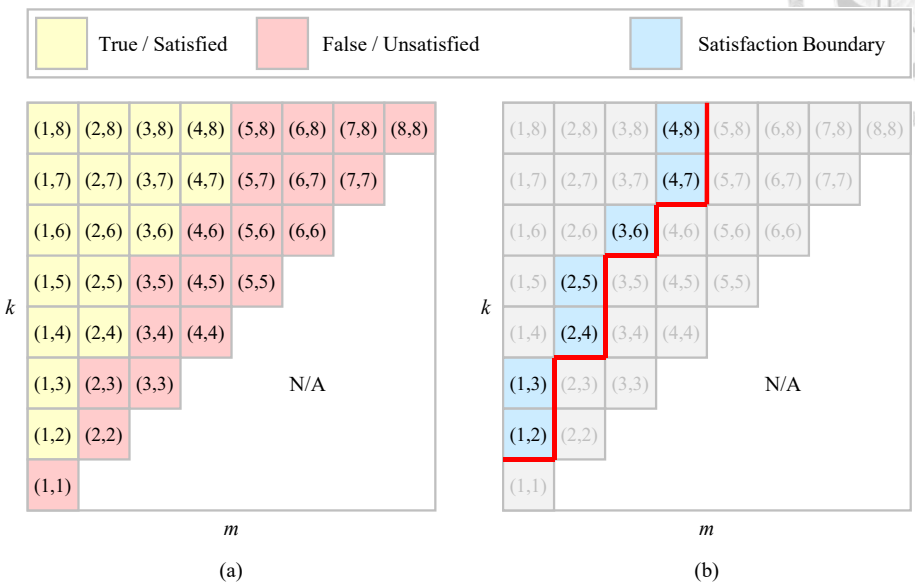


Figure 2.1: (a) An example safety table and (b) its satisfaction boundary.

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 be used to handle this special case situation).

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.

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

$$T[m,k] = \begin{cases} True & \text{if } m \le k \text{ and } \forall\sigma \models W(m,k),\ S \models P; \\ False & \text{if } m \le k \text{ and } \exists\sigma \models W(m,k),\ S \not\models P; \\ N/A & \text{if } m > k. \end{cases} \tag{2.1}$$

For $m > k$, $T[m,k]$ is not applicable as the corresponding weakly-hard fault model is undefined. Note that a safety table is computed off-line in design phase, and

doi:10.6342/NTU202200995

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 Figure 2.1(a).

Problem 1. ***Verification****. Given $K \in \mathbb{Z}^+$ and a transition system S, compute the safety table T.*

2.2 Weakly-Hard Scheduling

Definition 4. ***Task model*** *Each task τ_i is formally defined as (E_i, P_i, C_i), where*

- *E_i is the execution time of a job of task τ_i.*
- *P_i is the period of a job of task τ_i.*
- *C_i is the multiple weakly-hard constraints set of task τ_i.*

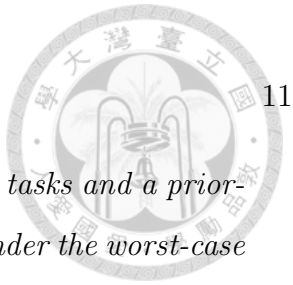
The multiple weakly-hard constraints set C_i can be represented as $C_i = \{(m_{i1}, k_{i1}), ..., (m_{in}, k_{in})\}$ where $\forall j \leq n, m_{ij} < k_{ij}$.

Each task in the taskset considers the latest job as the most useful one. That is, if a job misses its deadline, a system will discard the incomplete job and try to complete the latest one.

Definition 5. ***Priority Assignment Method****. A priority assignment method will assign the priority to each job for the task at each timestamp. The scheduling process at each timestamp will execute a job with the highest priority.*

A priority assignment method can be a fixed-priority method that fixes the priority for each task. It can also be a dynamic-priority method that dynamically adjusts the priority at each time stamp. Note that a job can face different priorities before completing by the definition.

doi:10.6342/NTU202200995



Problem 2. ***Schedulability Analysis***. *Given a taskset with N tasks and a priority assignment method, output whether the taskset is schedulable under the worst-case scenario.*

doi:10.6342/NTU202200995

Chapter 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).

Section 3.1 and 3.2 are the previous works refer to [23]. To introduce our further approaches and experimental results, we keep the notations and detailed algorithms for these sections.

3.1 Strength of Weakly-Hard Constraint

Definition 6. ***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 de-

doi:10.6342/NTU202200995

termine 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 $\sigma \models W(m',k)$. □

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.

doi:10.6342/NTU202200995

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
```

Definition 7. ***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 2.1(a) is shown in Figure 2.1(b). The reduction is crucial because we only need to store the satisfaction boundary rather than the whole safety table for the runtime monitor.

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 Chapter 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 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 3.1(a), if P is

doi:10.6342/NTU202200995

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

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 $\sigma \models W(xm, xk)$. □

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

doi:10.6342/NTU202200995

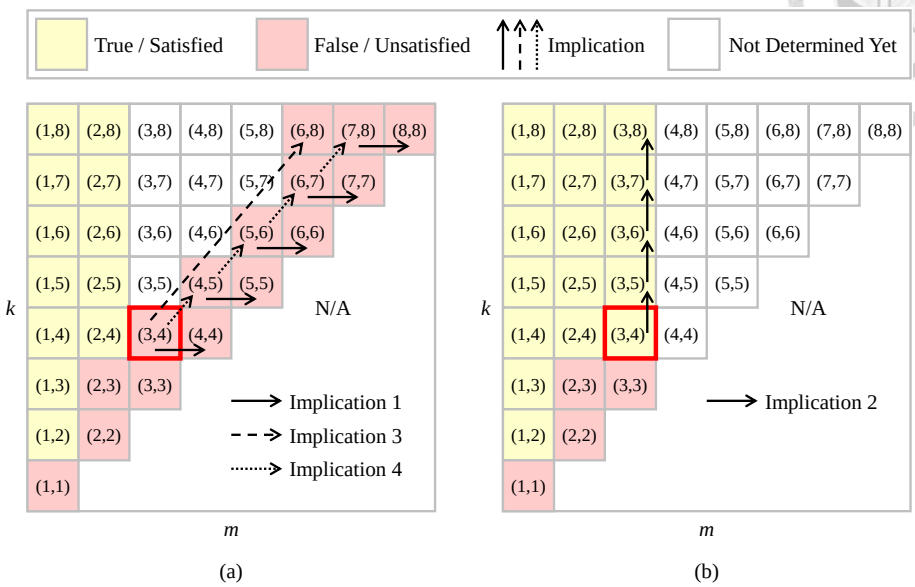


Figure 3.1: An illustration of Algorithms 1 (which applies Implications 1 and 2 only) and 2 (which applies Implications 1, 2, 3, and 4). To have a clear comparison, we focus on the implications of $W(3,4)$ only. (a) If P is unsatisfied under $W(3,4)$, then P is unsatisfied under $W(4,4)$. Algorithm 2 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 $k \geq 5$.

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 $\sigma \models W(m+x,k+x)$. □

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 $W(m,k)$.*

doi:10.6342/NTU202200995

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
```

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

doi:10.6342/NTU202200995

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
```

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 3.1(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 $m \geq k - 1$, 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 Figure 3.1(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

doi:10.6342/NTU202200995

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 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,

doi:10.6342/NTU202200995

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
```

and the system switches to a safe mode (Line 12). The runtime monitor then stores the input (Line 15) and continues monitoring.

doi:10.6342/NTU202200995

Chapter 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 Chapter 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 Chapter 2 where the satisfaction boundary is not given.

4.1 Definitions

Definition 8. ***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 9. ***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

doi:10.6342/NTU202200995

1, its complexity, or its runtime, we can define an optimal approach and an optimal verified set as follows:

Definition 10. ***Optimal Approach****. For any system and any property, an optimal approach computes the satisfaction boundary and minimizes the total verification cost.*

Definition 11. ***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 $W(1,3)$.

doi:10.6342/NTU202200995

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 Chapter 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
```

4.2 Optimal Verified Set Computation

Given the satisfaction boundary B, we propose Algorithm 5 to compute an optimal verified set, as Definition 11. 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 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 the 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

doi:10.6342/NTU202200995

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 12. ***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 13. ***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*

$$W(m,k) = W(m',k'), \text{ if and only if } S(W(m,k)) = S(W(m',k')).$$

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 $k = k'$. □

doi:10.6342/NTU202200995

Definition 14. ***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

doi:10.6342/NTU202200995

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 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 6, 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 $W(m_3,k_3)$. □

With Theorem 9, we can combine theorems and 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.

doi:10.6342/NTU202200995

- 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.

□

To this point, we prove 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.

doi:10.6342/NTU202200995

- 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. □

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.

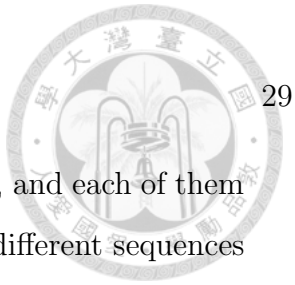
On the other hand, another implication is that, if $S(W(m,k)) \supseteq \bigcup_{i=1}^{n} S(W(m_i,k_i))$ and P is unsatisfied under at least one $W(m_i,k_i)$, then P is unsatisfied under $W(m,k)$. This implication can also be obtained from a single-constraint implication (from a constraint $W(m_i,k_i)$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

doi:10.6342/NTU202200995



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.

doi:10.6342/NTU202200995

Chapter 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 introduce a mask-compressing approach 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 Chapter 3. 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.

Section 5.2 and 5.3 are the previous works refer to [23]. In order to introduce section 5.4 and experimental results, we keep the notations and detailed algorithm for these sections.

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

doi:10.6342/NTU202200995

satisfying

$$\forall(q,x) \in Q \times \Sigma, \sum_{\overline{q} \in Q, (q,x,\overline{q}) \in \delta} P_r(q,x,\overline{q}) = 1, \tag{5.1}$$

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 introduce 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 Chapter 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.

doi:10.6342/NTU202200995

- There is a directed edge from $v_{q,mask}$ to $v_{\overline{q},\overline{mask}}$ if and only if

$$(q,\ \overline{mask}\ \%\ 2,\ \overline{q}) \in \delta, \tag{5.2}$$

$$(mask \cdot 2)\ \%\ 2^{k-1} + \overline{mask}\ \%\ 2 = \overline{mask}, \tag{5.3}$$

$$\text{Count1}(mask) + \overline{mask}\ \%\ 2 \leq m, \tag{5.4}$$

  where Count1() counts the number of 1's in a mask.

Equation (5.2) is for the transition in S, Equation (5.3) is for the 1-bit "shift" of the mask, and Equation (5.4) is for the number of 1's bounded by the weakly-hard fault model. 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 masks because we can achieve computationally efficient implementation by bit operations.

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 Chapter 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.

doi:10.6342/NTU202200995

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 (5.4), 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 thesis. 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 $G_{m-1} \subseteq G_m$.

doi:10.6342/NTU202200995

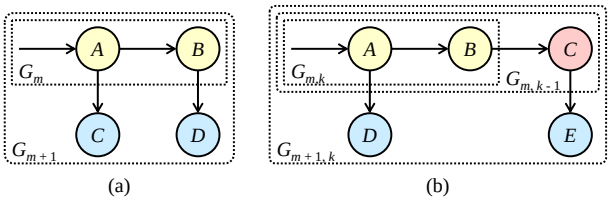


Figure 5.1: (a) An example layered BFS from $W(m,k)$ to $W(m+1,k)$. Vertices A and B are reachable in $W(m,k)$. Vertices C and D are unreachable in $W(m,k)$ but reachable in $W(m+1,k)$. (b) An example dual-layered BFS from $W(m,k)$ to $W(m,k-1)$ and then to $W(m+1,k)$. Vertices A and B are reachable in $W(m,k)$. Vertex C is unreachable in $W(m,k)$ but reachable in $W(m,k-1)$. Vertices D and E are unreachable in $W(m,k)$ and $W(m,k-1)$ but reachable in $W(m+1,k)$.

An example is shown in Figure 5.1, where vertices A and B are reachable (satisfying Equation (5.2)) and other vertices are unreachable (not satisfying Equation (5.2)). After performing the BFS for $W(m,k)$, we can collect a vertex set V'_m containing unreachable vertices (such as vertices C and D), and, for each $v' \in V'_m$, there exists a vertex $v \in V_m$ such that $(v,v') \in E_{m+1}$ (such as edges (A,C) and (B,D)). 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 $m' < m$.

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).

doi:10.6342/NTU202200995

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 $G_{m,k}$ and avoid redundancy as $G_{m,k} \subseteq G_{m,k-1}$. Similarly, upon performing the BFS on $G_{m+1,k}$, 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.

doi:10.6342/NTU202200995

Specifically, during the BFS on $G_{m,k}$, we collect two vertex sets $V'_{m,k-1}$ and $V'_{m+1,k}$, where, 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 $(v, v') \in E_{m,k-1}$; and 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 $(v, v') \in E_{m+1,k}$. 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 $V''_{m+1,k}$, where, for each $v'' \in V''_{m+1,k}$, there exists a vertex $v \in V_{m,k-1} - V_{m,k}$ such that $(v, v'') \notin E_{m,k-1}$ and $(v, v'') \in E_{m+1,k}$. 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 $B(k)$.

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, $B(k) = m$.

doi:10.6342/NTU202200995

Chapter 6

Schedulability Analysis

In this section, we consider the scheduling process. The consecutive events of a system will be formatted as a task τ_i with period P_i, execution time E_i, and weakly-hard constraints C_i. Given a taskset and a priority assignment method, the schedulability analysis proposed in this section will analyze whether the taskset is schedulable or not considering the worst-case scenario. Section 6.1 introduces the job trace model which represents the status of a task. Section 6.2 and 6.3 exploit the job trace models of the taskset to compute the worst-case response time. Section 6.4 further reconstructs the job trace models and gives the results to the schedulability analysis.

6.1 Job Trace Model

Definition 15. ***Job Trace****. For a task τ_i at a certain timestamp, the job trace t is a binary sequence with length K_i which is the past K_i jobs status where $K_i = \max(k_{i1}, ..., k_{in})$.*

Job trace is a representation of the task status. For example, if the results of the past three jobs are miss, miss, and meet, the job trace at this timestamp will be 110. The reason why we need to define the length of the job trace K_i is that the resulting job trace can also represent whether the weakly-hard constraints are satisfied or not.

doi:10.6342/NTU202200995

Definition 16. ***Job Trace Model***. *The job trace model of a task τ_i can be represented as a directed graph $G_i = \{V_i, E_i\}$, where*

- *V_i is the set of all possible job traces of task τ_i.*
- *E_i is the set of edges which represent the deadline meet or miss.*

The edge represents whether the deadline of the current job is met or missed. That is, if the current job of a job trace t meets the deadline, we will have an edge from t to t' where $t' = (t << 1) \wedge 0$. For example, if the current job of a job trace 111 meets the deadline, we will have an edge from 111 to 110. Initially, if we consider all possible job trace that a task can face, there will be 2_i^K vertices and 2^{K_i+1} edges.

The reason why we need the job trace model is that the job trace model represents all job trace a task will face, and the job trace has the directed relation to the weakly-hard constraint. However, during the scheduling process, there might be some job trace in the job trace model which is impossible to reach. Therefore, the rest of the schedulability analysis will focus on removing those job traces in the job trace model base on the worst-case scenario.

6.2 Maximum Interrupt Jobs

The computation of the maximum interrupt jobs can help analyze the worst-case response time. Given a job trace t of a task τ_i, another task τ_j, and a number n, we target to know the maximum number of the jobs with higher priority within n consecutive jobs of task τ_j. Note that the priority assignment method can map one job trace to one priority, and the higher priority here means the priority p' of the job trace of task τ_j is higher than the priority p of the job trace t.

doi:10.6342/NTU202200995

Algorithm 6 Maximum Interrupt Jobs Computation

```
1: procedure COMPUTE_MAXIMUM_INTERRUPT_JOBS(p, V_j, E_j, PM, n)
2:     M ← [ ]
3:     if any v ∈ V_j, PM(v) > p then        ▷ Obtain the maximum interrupt jobs within 1 consecutive job
4:         M[1] ← 1
5:     else
6:         M[1] ← 0
7:     end if
8:     A ← adjacency list of V_j, E_j
9:     B ← [ ]
10:    for v ∈ V_j do        ▷ Assign 0 or 1 to each vertex according to the priority
11:        if PM(v) > p then
12:            B[v] ← 1
13:        else
14:            B[v] ← 0
15:        end if
16:    end for
17:    for k ← 2 to n do
18:        NB ← [ ]
19:        for v ∈ V_j do
20:            for v' ∈ A[v] do        ▷ Broadcast the temporary maximum interrupt jobs to its adjacency vertices
21:                if PM(v') > p then
22:                    NB[v'] ← Max(NB[v'], B[v] + 1)
23:                else
24:                    NB[v'] ← Max(NB[v'], B[v])
25:                end if
26:            end for
27:            M[k] ← Max(NB[v], v ∈ V_j)
28:            B ← NB
29:        end for
30:    end for
31:    return M
32: end procedure
```

To make it more convenient for the worst-case response time analysis, given n, Algorithm 6 will output the maximum interrupt jobs within 1 to n consecutive

doi:10.6342/NTU202200995

jobs and store them in B. At k^{th} iteration (Line 17), each vertex will consider itself as the end of the $k-1$ consecutive jobs and store the temporary maximum interrupt jobs. Next, each vertex will broadcast its temporary maximum interrupt jobs to its adjacency vertices (Line 20) and update them with the interrupt jobs of k consecutive jobs (Lines 21–25). For each iteration, the maximum interrupt jobs of k consecutive jobs will be the maximum value among all vertices (Line 27).

The time complexity of Algorithm 6 will depend on n and the number of vertices and edges. For each iteration (Line 17), the broadcast method of the job trace model will cost $O(|V_j| + |E_j|)$. Consider the initial job trace model, the time complexity of Algorithm 6 will be $O(n \times 2^{K_j})$.

6.3 Worst-Case Response Time

For each job trace t with priority p of a task τ_i, we target to find the worst-case response time, where every other task τ_j faces the maximum interrupt jobs. Based on the maximum interrupt jobs computation, we can compute the temporary interference from other tasks and recursively compute the worst-case response time.

Theorem 17. *Given a time window w, the worst-case temporary interference of a job trace t with priority p of a task τ_i will be*

$$I(t, \tau_i, w) = \sum_{\tau_j \neq \tau_i} M^p_{\tau_i,\tau_j} \left\lceil \left\lceil \frac{w}{P_j} \right\rceil \right\rceil \times E_j. \tag{6.1}$$

$\left\lceil \frac{w}{P_j} \right\rceil$ represents the number of the consecutive jobs of task τ_j during time window w. Since we have obtain $M^p_{\tau_i,\tau_j}$ by the Algorithm 6 beforehand, we just need to summarize the maximum interrupt jobs of all the other tasks.

Theorem 18. *The worst-case response time of a job trace t with priority p of a*

doi:10.6342/NTU202200995

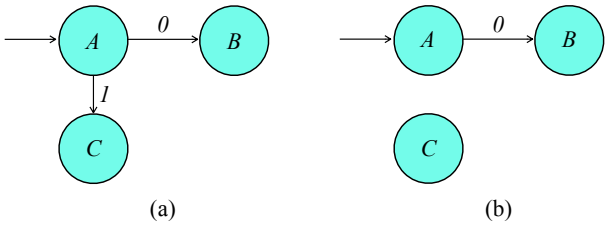


Figure 6.1: (a) The part of the job trace model. (b) Since the worst-case response time of the job trace A is less than the period, it represents the resulting job trace model after removing edge $\overline{AC}$

task τ_i will be computed by the following recurrence

$$W(t, \tau_i, m+1) \leftarrow E_i + I(t, \tau_i, W(t, \tau_i, m)). \tag{6.2}$$

The recurrence is initialized with $W(t, \tau_i, 0) \leftarrow E_i$ and terminates when $W(t, \tau_i, m) > P_i$ or $W(t, \tau_i, m+1) = W(t, \tau_i, m)$.

When $W(t, \tau_i, m) > P_i$, the current job of τ_i might possibly miss base on the worst-case scenario. That is, we do not need to continue the recurrence. The time complexity of the recurrence will be $O(|\tau| \times n \times 2^{K_j} + r \times |\tau|)$, where $|\tau| \times n \times 2^{K_j}$ denotes the preprocessing of the maximum interrupt jobs and $r \times |\tau|$ denotes the recurrence. Note that r is the recursive calls, and the maximum n will be $\left\lceil \frac{P_{max}}{P_{min}} \right\rceil$, where P_{max}, P_{min} are the maximum and minimum period of the taskset.

6.4 Job Trace Model Reconstruction

After computing all the worst-case response times of every task, we target to remove vertices and edges from the job trace model. Since the job trace model represents all possible statuses a task can reach, if the worst-case response time of a job trace is less or equal to the period, the job with the job trace will meet the deadline certainly, and we can further remove the missed edge of the job trace.

doi:10.6342/NTU202200995

Algorithm 7 Job Trace Model Stabilization

```
1: procedure JOB_TRACE_MODEL_STABILIZATION(τ)
2:     Initialize job trace models with τ.
3:     do
4:         for every job trace do
5:             Compute maximum interrupt jobs
6:             Compute worst case response time
7:         end for
8:         for every job trace do
9:             if worst case response time < period then
10:                Remove the missed edge
11:            end if
12:        end for
13:    while Being able to remove any edge from job trace models
14: end procedure
```

For example, in Figure 6.1, after the worst-case response time computation, the worst-case response time of the job trace A is less than the period. That is, the job with the job trace A will always meet the deadline, and we can remove the missed edge $\overline{AC}$. If every inbound of C is removed, C can further be removed.

We introduced Algorithm 7 to recursively remove the missed edge of the job trace models. Since the removing (Line 10) will lead to the change of the worst-case response time computation (Lines 5–6), the recursive method will make the worst-case response time tighter.

Assume that the priority assignment method will give q kinds of priority to the taskset, and the time complexity for the job trace model stabilization will be $O(s \times q \times |\tau| \times n \times 2^{K_j})$. Note that s is the number of recursive calls (Line 13).

Theorem 19. ***Schedulability Analysis*** *If all of the job trace satisfy the weakly-hard constraints after stabilization, the taskset will be schedulable in any case.*

Despite the introduced schedulability analysis being general to the priority

doi:10.6342/NTU202200995

assignment method, the priority assignment method should follow that one job trace can only be mapped to one priority. If not, we could construct another one-to-one mapping by considering another status instead of considering past K_i jobs status.

doi:10.6342/NTU202200995

Chapter 7

Experimental Results

7.1 Weakly-Hard Verification

To compare the efficiency of different approaches, we implemented a brute-force approach which 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 $C(K)$.

7.1.1 Discrete Second-Order Control

7.1.1.1 Setting

The case study is a discrete second-order controller under perturbation attacks. The objective of the controller is to maintain the position at a fixed value (0 in our case), and the attacker attempts to shift the position away from the fixed value. The detailed configuration can refer to [23]. The unsafe state represents the state where the control position is out of the range. Verifying whether the control

doi:10.6342/NTU202200995

Table 7.1: 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

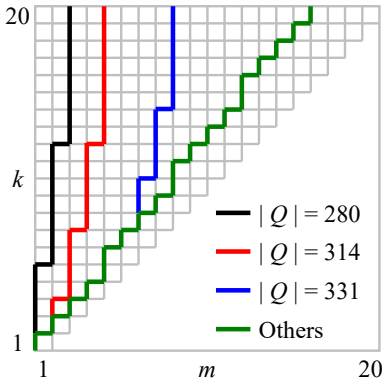


Figure 7.1: Discrete second-order control: computed satisfaction boundaries.

position is in the safe range under perturbation attacks is reduced to solving for the reachability of the unsafe state for the finite-state machine.

7.1.1.2 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 $\dot{x}_{\min} = -4$, $\dot{x}_{\max} = 4$, $\ddot{x}_{\mathrm{C}} = 2$, and $S_{\mathrm{atk}} = \{5\}$ 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.

doi:10.6342/NTU202200995

Table 7.2: 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

The results are shown in Table 7.1, and the corresponding satisfaction boundaries are illustrated in Figure 7.1, 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

doi:10.6342/NTU202200995

heuristic. Moreover, the dual-layered BFS approach mostly has the best efficiency among all other approaches except the optimal approach.

7.1.1.3 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 7.2, 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 properties are 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.

7.1.2 Network Routing

7.1.2.1 Setting

The case study is network routing of Extranet. There are two routing paths with the same source and destinations on one router. We denote the delay levels

doi:10.6342/NTU202200995

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

$$l_1(t{+}1) \leftarrow l_1(t){+}(2{\cdot}d(t){-}1);\;\; w_2(t{+}1) \leftarrow w_2(t){+}1;\;\; l_2(t{+}1) \leftarrow 0;\;\; w_1(t{+}1) \leftarrow 0, \tag{7.1}$$

  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.

doi:10.6342/NTU202200995

- If $s(t) = 1$, meaning that the second routing path is in use (switched from the first routing path), then

$$l_2(t{+}1) \leftarrow l_2(t){+}(2{\cdot}d(t){-}1);\ \ w_1(t{+}1) \leftarrow w_1(t){+}1;\ \ l_1(t{+}1) \leftarrow 0;\ \ w_2(t{+}1) \leftarrow 0, \tag{7.2}$$

  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.*,

$$s(t+1) \leftarrow \begin{cases} 1, & \text{if } s(t) = 0, l_1(t) > \tau_1, w_2(t) > \gamma; \\ 0, & \text{if } s(t) = 1, l_2(t) > \tau_2, w_1(t) > \gamma; \\ s(t), & \text{otherwise.} \end{cases} \tag{7.3}$$

- 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.*,

$$c(t+1) \leftarrow \begin{cases} 1, & \text{if } s(t) = 0, l_1(t) > \tau_1, w_2(t) \leq \gamma; \\ 1, & \text{if } s(t) = 0, l_2(t) > \tau_2, w_1(t) \leq \gamma; \\ 0, & \text{otherwise.} \end{cases} \tag{7.4}$$

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 $d(t)$.

doi:10.6342/NTU202200995

Table 7.3: 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 7.4: 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

- 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.

7.1.2.2 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

doi:10.6342/NTU202200995

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 7.3, 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 10 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.

7.1.2.3 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 7.4, 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.

7.1.3 Lane Changing

7.1.3.1 Setting

The case study is lane changing with two acceleration controllers on two vehicles driving on two lanes along a road segment with length $x_{\max}$. We denote the

doi:10.6342/NTU202200995

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

$$x(t+1) \leftarrow \min\left(x(t) + v(t) + \frac{1}{2} \cdot a(t), x_{\max}\right), \tag{7.5}$$

$$v(t+1) \leftarrow \max\left(\min\left(v(t) + a(t), v_{\max}\right), 0\right), \tag{7.6}$$

doi:10.6342/NTU202200995

$$a(t+1) \leftarrow \begin{cases} 0, & \text{if } s(t) = 0; \\ a_{\min}, & \text{if } s(t) = 1,\ |x'(t) - x(t)| < 2l,\ v(t) < v'(t); \\ a_{\max}, & \text{if } s(t) = 1,\ |x'(t) - x(t)| < 2l,\ v(t) \geq v'(t); \\ a(t), & \text{otherwise.} \end{cases} \tag{7.7}$$

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

$$x'(t+1) \leftarrow \min\left(x'(t) + v'(t) + \frac{1}{2} \cdot a'(t), x_{\max}\right), \tag{7.8}$$

$$v'(t+1) \leftarrow \max\left(\min\left(v'(t) + a'(t), v_{\max}\right), 0\right), \tag{7.9}$$

$$a'(t+1) \leftarrow \begin{cases} a_{\max}, & \text{if } s'(t) = 0; \\ a_{\min}, & \text{if } s'(t) = 1,\ |x(t) - x'(t)| < 2l,\ v(t) \geq v'(t); \\ a_{\max}, & \text{if } s'(t) = 1,\ |x(t) - x'(t)| < 2l,\ v(t) < v'(t); \\ a'(t), & \text{otherwise.} \end{cases} \tag{7.10}$$

We also introduce the following transition function:

$$c(t+1) \leftarrow \begin{cases} 1, & \text{if } c(t) = 1; \\ 1, & \text{if } c(t) = 0, x(t) \neq x_{\max} \text{ or } x'(t) \neq x_{\max},\ |x'(t) - x(t)| \geq 2l; \\ 0, & \text{otherwise.} \end{cases} \tag{7.11}$$

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

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

doi:10.6342/NTU202200995

Table 7.5: 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 7.6: 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

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.

7.1.3.2 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 $v_{\max} = v'_{\max} = 10, a_{\max} = a'_{\max} = 5, a_{\min} = a'_{\min} = -5, l = 4$ 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 7.5, 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

doi:10.6342/NTU202200995

most cases. Similarly, an optimal approach defined in Definition 10 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.

7.1.3.3 Experiment on K

We experimented on how each approach scales with respect to K. We fixed $x_{\max} = 100, v_{\max} = v'_{\max} = 10, a_{\max} = a'_{\max} = 5, a_{\min} = a'_{\min} = -5, l = 4$. The results are shown in Table 7.6, 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.

7.1.4 Summary

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 suitable for general properties and general systems.

7.2 Weakly-Hard Scheduling

In this section, we compare our schedulability analysis method with another method. Different methods have different worst-case scenarios, time complexity, and scalability. To compare different methods, the first part of this section will

doi:10.6342/NTU202200995

target the taskset generation, and the second part will apply different methods to the taskset and specify the advantage of our method.

7.2.1 Taskset Generation

In order to compare different schedulability analysis methods, the taskset generation is needed. For a target total utilization U, we generate 100 tasksets. The sum utilization of the tasks for a taskset will be U. For a taskset, the task utilization is assigned uniformly by the UUniFast algorithm [4]. The period of the task is assigned uniformly in $[10, 100]$. We will have 3 weakly-hard constraints.

k of the weakly-hard constraint is assigned uniformly in $[5, 15]$, and m is assigned uniformly in $[2, k)$. Note that each two weakly-hard constraints should be incomparable by 7, and 8. If two weakly-hard constraints are comparable, the trace set of one constraint will be the subset of the other. That is, we only need to consider one constraint instead of multiple constraints.

7.2.2 Schedulability Analysis Comparison

We compare our schedulability analysis (JTMS) to an existing method (JCLS). JTMS represents our job trace model stabilization method and JCLS represents the job-class-level scheduler with reachability tree analysis method [5]. For each taskset, the method will output whether the taskset is schedulable or not. For a target total utilization U, the method will output a floating value denoting the ratio of the schedulable tasksets among 100 tasksets. If a method has a higher ratio for a target U, the worst-case analysis of the method has a tighter upper bound. For example, if a taskset is schedulable with method A, but it is unschedulable with method B, the worst-case considered by method B will not occur during scheduling.

We apply job-class-level fixed priority scheduling priority as our priority as-

doi:10.6342/NTU202200995

Table 7.7: Schedulability Analysis Comparison: schedulability ratio with different values of U.

U	1.0	1.1	1.2	1.3
JTMS	0.58	0.44	0.31	0.28
JCLS	0.58	0.44	0.31	0.28

signment method [5]. However, the method can only deal with a single weakly-hard constraint. For multiple weakly-hard constraints, we obtain the priority with the constraint with the highest ratio and consider the condition of multiple weakly-hard constraints.

Table 7.7 stores the schedulability ratio of different methods and different U. The results of JTMS and JCLS become the same, which denotes that the worst-case analysis of these two methods leads to a similar worst-case response time while the worst-case scenarios of these two methods are different. We have further analyzed the worst-case response time of JTMS and JCLS and found that jobs with the same status have similar worst-case response times for these two methods. For the scalability, JTMS can support any kind of priority assignment method if the method meets the one-to-one condition while JCLS can only support the job-class-level fixed priority scheduling priority. However, JTMS has a higher time complexity than JCLS.

doi:10.6342/NTU202200995

Chapter 8

Conclusions

In this thesis, 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 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.

Moreover, considering multiple systems sharing a processor, a scheduler is needed. We studied the multiple weakly-hard constraints scheduling problems and proposed the generalized schedulability analysis method to obtain whether the taskset is schedulable or not beforehand. For future directions, system designers can come up with different strategies to assign the weakly-hard constraints for systems, or the scheduler can exploit different priority assignment methods. Both of them can bring benefits to the scheduling process. Last but not least, the proposed

doi:10.6342/NTU202200995

schedulability analysis has a higher time complexity than others which can further be optimized.

The future works of the verification and schedulability analysis include:

- Consider multiple weakly-hard constraints simultaneously for a system which could lead to higher dimensional safety table.
- Design a more robust priority assignment method for the weakly-hard scheduling problem.
- Design a strategy to assign the weakly-hard constraints for systems which can bring benefits to the weakly-hard scheduling problem.
- Extend the scheduling problem to other constraints. A simple method can be considering the composition of different constraints to be the status of the job trace model.
- Extend the scheduling problem with the multiple processors scenario. A simple method can be considering jobs with top n priorities when there are n processors.

doi:10.6342/NTU202200995

Bibliography

[1] L. Ahrendts, S. Quinton, T. Boroske, and R. Ernst, "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] G. Bernat, A. Burns, and A. Liamosi, "Weakly hard real-time systems," *IEEE Transactions on Computers*, vol. 50, no. 4, pp. 308–321, 2001.

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

[4] E. Bini, "Measuring the performance of schedulability tests," *Real-Time Systems*, vol. 30, pp. 129–154, 05 2005.

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

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

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

doi:10.6342/NTU202200995

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

[9] A. Gujarati, M. Nasri, R. Majumdar, and B. Brandenburg, "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] M. Hamdaoui and P. Ramanathan, "A dynamic priority assignment technique for streams with (m, k)-firm deadlines," *IEEE Transactions on Computers*, vol. 44, no. 12, pp. 1443–1451, 1995.

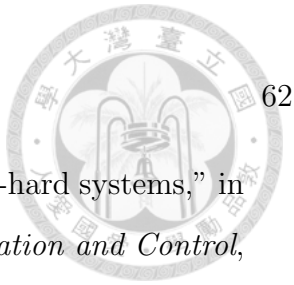
[11] Z. A. H. Hammadeh, R. Ernst, S. Quinton, R. Henia, and L. Rioux, "Bounding deadline misses in weakly-hard real-time systems with task dependencies," in *Design, Automation and Test in Europe Conference*, pp. 584–589, 2017.

[12] Z. A. H. Hammadeh, S. Quinton, M. Panunzio, R. Henia, L. Rioux, and R. Ernst, "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] Z. Hammadeh, S. Quinton, and R. Ernst, "Extending typical worst-case analysis using response-time dependencies to bound deadline misses," 10 2014.

[14] C. Huang, K.-C. Chang, C.-W. Lin, and Q. Zhu, "Saw: A tool for safety analysis of weakly-hard systems," in *Computer Aided Verification*, S. K. Lahiri and C. Wang, Eds., pp. 543–555. Cham: Springer International Publishing, 2020.

doi:10.6342/NTU202200995



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

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

[17] J. Li, Y. Song, and F. Simonot-Lion, "Providing real-time applications with graceful degradation of QoS and fault tolerance according to (m, k)-firm model," *IEEE Transactions on Industrial Informatics*, vol. 2, no. 2, pp. 112–119, 2006.

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

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

[20] S. Quinton, M. Hanke, and R. Ernst, "Formal analysis of sporadic overload in real-time systems," in *2012 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 515–520, 2012.

[21] S. Quinton, M. Negrean, and R. Ernst, "Formal analysis of sporadic bursts in real-time systems," pp. 767–772, 01 2013.

doi:10.6342/NTU202200995

[22] Y. Sun and M. Di Natale, "Weakly hard schedulability analysis for fixed priority scheduling of periodic real-time tasks," *ACM Transactions on Embedded Computing Systems*, vol. 16, no. 5s, pp. 171:1–171:19, 2017.

[23] S.-L. Wu, C.-Y. Bai, K.-C. Chang, Y.-T. Hsieh, C. Huang, C.-W. Lin, E. Kang, and Q. Zhu, "Efficient system verification with multiple weakly-hard constraints for runtime monitoring," in *Runtime Verification*, J. Deshmukh and D. Ničković, Eds., pp. 497–516. Cham: Springer International Publishing, 2020.

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

doi:10.6342/NTU202200995