Certified Tools for Schedulability Analyses

Xiaojie Guo

▶ **To cite this version:**

Xiaojie Guo. Certified Tools for Schedulability Analyses. Other [cs.OH]. Université Grenoble Alpes [2020-..], 2020. English. NNT : 2020GRALM073 . tel-03227425

HAL Id: tel-03227425
https://theses.hal.science/tel-03227425

Submitted on 17 May 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ GRENOBLE ALPES

Spécialité : Informatique

Arrêté ministériel : 25 mai 2016

Présentée par

Xiaojie GUO

Thèse dirigée par **Pascal FRADET**, Université Grenoble Alpes
et codirigée par **Sophie QUINTON**, INRIA
et **Jean-François MONIN**, professeur Polytech Grenoble, UGA

préparée au sein du **Laboratoire d'Informatique de Grenoble**
dans **l'École Doctorale Mathématiques, Sciences et technologies de l'information, Informatique**

Outils certifiés pour les analyses d'ordonnançabilité

Certified Tools for Schedulability Analyses

Thèse soutenue publiquement le **18 décembre 2020**,
devant le jury composé de :

Monsieur PASCAL FRADET
CHARGE DE RECHERCHE HDR, INRIA CENTRE DE GRENOBLE RHÔNE-ALPES, Directeur de thèse
Monsieur ROBERT DAVIS
CHERCHEUR SCIENTIF. SENIOR EQUIV. HDR, UNIVERSITE D'YORK -ROYAUME-UNI, Rapporteur
Monsieur STEPHAN MERZ
DIRECTEUR DE RECHERCHE, INRIA CENTRE NANCY GRAND EST, Rapporteur
Monsieur NICOLAS NAVET
PROFESSEUR, UNIVERSITE DU LUXEMBOURG, Examinateur
Monsieur DAVID MONNIAUX
DIRECTEUR DE RECHERCHE, CNRS DELEGATION ALPES, Président

Abstract

Schedulability analysis aims at guaranteeing the absence of deadline misses in hard real-time systems. This property is crucial for systems used in safety-critical domains such as avionics because a single deadline miss may have catastrophic consequences. In this thesis, we use the Coq proof assistant and machine-checked proofs to provide the highest level of confidence in hard real-time systems schedulability analyses as well as related industrial tools.

The main contributions of this thesis are:

(i) A formal interface combining the schedulability analyses proven in the Prosa library based on Coq with a formally verified OS kernel (RT-CertiKOS). This work demonstrated the adequacy of Prosa's abstract model with a real system and showed that analyses proven over an abstract and analysis-convenient model can be also applied to a concrete system. This work also provided RT-CertiKOS with a modular, state-of-the-art schedulability analysis proof.

(ii) CertiCAN, a formally verified result certifier for CAN (Controller Area Network) analyzers. This work showed that result certification is a flexible and light-weight process suitable for industry practice. Indeed, it does not rely on the source code and is not impacted by software updates. Our experiments show that CertiCAN is efficient enough to certify the results produced by the industrial tool RTaW-Pegase even for large systems.

(iii) GD, a very general task model and its corresponding response time analysis amenable to a formalization in Coq. The main benefit of this approach is to factorize and reduce the proof effort. After verifying a general schedulability analysis for GD, proving the correctness of an analysis for a more specific model boils down to proving its instantiation to GD.

Résumé

L'analyse d'ordonnancabilité vise à garantir le respect des échéances dans les systèmes temps réel durs. Cette propriété est cruciale pour les systèmes utilisés dans les domaines critiques tels que l'avionique, car une échéance manquée peut avoir des conséquences catastrophiques. Dans cette thèse, nous utilisons l'assistant de preuves Coq afin d'assurer la correction des analyses d'ordonnancabilité des systèmes temps réel durs et des outils industriels associés.

Les principales contributions de cette thèse sont:

(i) Une interface formelle combinant les analyses d'ordonnancabilité prouvées dans la bibliothèque Prosa basée sur Coq avec un noyau de sytème d'exploitation concret vérifié formellement (RT-CertiKOS). Ce travail a permis de justifier l'adéquation du modèle abstrait de Prosa à un système réel. Il a montré que les analyses prouvées pour un modèle abstrait et dédié à l'analyse peuvent également être appliquées à un système concret. Ce travail a également fourni à RT-CertiKOS une preuve d'ordonnancabilité modulaire à la pointe de l'état de l'art.

(ii) CertiCAN, un certificateur de résultats formellement vérifié pour les analyseurs de réseaux CAN (Controller Area Network). Ce travail a montré que la certification de résultats est un processus flexible et léger qui convient bien aux pratiques de l'industrie. En effet, CertiCAN n'a pas besoin d'avoir accès au code source et n'est pas affecté par les mises à jour logicielles. Nos expérimentations ont montré que CertiCAN est suffisamment efficace pour certifier les résultats produits par l'outil industriel RTaW-Pegase et ceci même pour de grands systèmes.

(iii) GD, un modèle de tâches très général et une analyse du temps de réponse associée se prêtant à sa formalisation en Coq. L'avantage de cette aproche est de factoriser et de simplifier l'effort de preuve. Une fois l'analyse du temps de réponse pour GD formellement vérifiés, prouver la correction d'une analyse pour un modèle plus spécifique revient à prouver son instanciation à GD.

Acknowledgements

Firstly, I would like to express my deepest gratitude to my advisors Pascal Fradet, Jean-François Monin, and Sophie Quinton for their support, encouragement, and patience during the past four years. Without their guidance and support this thesis would not have been possible.

I would also like to extend my appreciation to my committee: Robert Davis and Stephan Merz for taking time to read my thesis, writing the evaluation report, and giving me a lot of valuable advices and comments that have helped me to improve this thesis; Nicolas Navet and David Monniaux for their insightful questions and comments during my Ph.D defense.

I had great pleasure of working with Lionel Rieg, Maxime Lesourd, Mengqi Liu, and Zhong Shao. Particular thanks to them for their help and contributions. I really appreciate Zhong Shao for inviting me for a short visit to his team.

I am grateful to my teams: PACCS team at Verimag and SPADES team at Inria Grenoble for plenty of interesting seminars and discussions. Many thanks to my colleagues for their helpful comments and suggestions on my work. Many thanks to Helen for helping me with many administrative procedures.

Special thanks to the LabEx Persyval-Lab and the RT-Proofs project for the financial support and RealTime-at-Work for granting me with an academic license for RTaW-Pegase.

Finally, I am indebted to my parents, my sister, and my wife for their continuous love, unwavering support, and encouragement during my Ph.D work.

Contents

List of Figures

Contents

Chapter 1

Introduction

Embedded systems are ubiquitous in our daily lives. They are widely used in automobiles, avionics, telecommunications, medical devices, and many other domains. Given that embedded systems are typically designed for reacting with a physical environment that is continuously evolving, they are expected to respond to that environment in real-time, that is, in a timely manner. For instance, in a car system, after the brake pedal is pressed, actual braking must take place within a few milliseconds. Such systems are called *real-time systems*. Real-time systems are thus required to satisfy both functional and temporal properties.

In the past decades, the real-time community has established a huge body of research on real-time systems analysis in order to ensure temporal properties. Most published works establish correctness of the proposed analyses using *pen-and-paper* proofs which often depend on human intuitions, implicit assumptions and omit many details. They do not provide *formal* guarantees about the results of the analyses.

The goal of this thesis is to promote formal methods for real-time systems analysis in order to provide high confidence in their results. To motivate our research, we start with an introduction to critical real-time systems, their analysis and formal verification. We then state the research problems tackled in this thesis, summarize our contributions, and present the outline of this document.

Contents

1.1 Critical Real-Time Systems

A system is said to be *real-time* when its responses to a stimulus should satisfy some temporal constraints. For instance, an airplane pilot system should respond to the pilot's instructions quickly and predictably, otherwise there may be catastrophic consequences. Such temporal constraints are usually specified as *deadlines*: the response of a real-time system to some stimulus is expected to meet its deadline, that is, to happen before a specified maximum delay. For different real-time systems, deadline misses will have different consequences. According to the criticality of consequences, real-time systems can be classified into three categories (more details in Section 2.2): hard real-time systems, weakly hard (or firm) real-time systems, and soft real-time systems.

In this thesis, we focus on hard real-time systems where missing one deadline can lead to system failure and may have catastrophic consequences. Such systems have attracted a lot of attention in the community due to their utmost criticality, especially for those used in safety-critical domains such as automotive and avionics.

A real-time system consists of two parts:

- **Software.** System functionalities are typically implemented as software programs that can be decomposed into several *tasks*. The more system functionalities there are, the higher the number of tasks and system complexity.
- **Hardware.** The relevant hardware elements for real-time systems are processor units to execute the code instructions, memories to store data and programs, and communication buses to connect system components for a coordinated manipulation. The complexity of systems highly depends on the type of processors and bus architectures. For instance, a uniprocessor executes only one task at a time, while a multiprocessor can execute several tasks simultaneously and may even allow task migrations between processors. In automotive domain, Electronic Control Units (ECUs) are distributed in different parts of a vehicle, which are connected by buses for transmitting messages between them according to some protocols *e.g.,* CAN (Controller Area Network). Each ECU uses its own time clock to communicate with the others, which increases the complexity of analyses (see Chapter 4).

Hard real-time systems can be very complex. In particular, in the automotive domain and avionics, they have to sustain not only basic system controls but also increasing functionalities.

In the automotive domain, a car should manage many functional activities all together such as engine control, electric throttle control, battery management, transmission control, navigation, airbag deployment and so on. Nowadays, with the development of autonomous driving, car systems are becoming even more complex in order to meet other functionalities such as connectivity, automation and safety. All these functionalities are implemented as software applications, which are executed on ECUs. In a modern Volvo vehicle (in 2020), there are more than 120 ECUs connected to react to about 7000 external signals. These functionalities are implemented by 100 million lines of code (LOC) [9].

In avionics, aircraft systems have evolved over several decades, from realizing the basic functionalities such as flight control, hydraulics and electrical functions in the early systems to incorporate many additional functionalities such as autopilot, cooling system, optimization of flight plans, reduction of fuel consumption, air traffic management, anti-collision system and in-flight entertainment. In 2005, there were already 100000 functionalities implemented using millions of LOC in an aircraft system [76]. For instance, Boeing 787 contains about 7 million LOC [109].

As the system complexity increases, it is more and more difficult to validate temporal correctness. For safety-critical real-time systems, it is not sufficient to validate the temporal correctness by testing because this does not cover corner cases and one deadline miss may cause a loss of lives. It is therefore important to provide formal guarantees that there is no deadline miss in hard real-time systems.

1.2 Schedulability Analyses and Their Correctness

Schedulability analysis aims at guaranteeing the absence of deadline misses for hard real-time systems. In the literature, many schedulability analyses have been proposed since Liu and Layland's seminal work [96]. The correctness of most published analyses is established using *pen-and-paper* proofs, which rely on implicit assumptions and human intuitions and sometimes omit many details. As a result, the proofs are difficult to check, generalize and reuse because they do not point out which implicit assumptions are really needed and do not address the correctness of intuitions and omissions. In fact, quite a few analyses have been proven faulty [48, 43, 21]. In particular, the original schedulability analysis [144, 147, 145] for CAN buses has been shown to be flawed [25] and was corrected more than ten years later [43]. Before the revised version, the original analysis has been recognized incorrect by Volvo Car Corporation and used in the configuration and analysis of CAN buses

for Volvo S80 [28]. This illustrates the lack of formal guarantees for analysis results and motivates the use of formal methods for providing higher confidence in these analyses.

Formal verifications such as *model checking* and *theorem proving* are mathematical techniques used to provide formal guarantees for software systems, analyses, or mathematical theorems. Model checking has been widely and successfully used in industrial applications. However, it suffers from a state-space explosion problem and does not scale to large and complex systems without proper abstraction techniques. In contrast to model checking, theorem proving (in particular interactive theorem proving) is able to prove arbitrarily complex problems and to mechanically check proofs. The advent of recent mechanized proofs projects such as seL4 [80], CompCert [91], CertiKOS [66] shows that theorem proving techniques and tools (in particular the Coq proof assistant) have now reached a maturity level such that they can be used to formally verify schedulability analyses.

Both industry practice and academic research advocate the use of formal verification in order to provide the highest level of confidence in industrial applications and academic results. In industry practice, there already exist international security and safety assurance certification standards such as Common Criteria, ISO26262 for the automotive domain and DO-178C for avionics, which recommend the use of formal verifications for the development and validation of safety-critical real-time systems. For these standards, formal verifications achieve the highest security and safety assurance level for industry tools. In academic research, applying theorem proving to formally verify the correctness of real-time systems analyses has the added benefit that it helps to get a better understanding of the role played by assumptions and to generalize results. More importantly, it rules out flaws in the corresponding proofs which have been mechanically checked.

The goal of this thesis is to use theorem proving (specifically the Coq proof assistant) to formally specify and verify schedulability analyses. This takes place within the Prosa project and library [112], a machine-checkable framework for schedulability analysis using Coq.

1.3 Research Problems

Applying theorem proving to the formal verification of schedulability analyses for hard real-time systems gives rise to several research problems:

1. How can we justify the adequacy of system definitions specified using a theorem prover? In fact, these definitions are usually formalized in a very abstract

manner, which are suitable for reasoning about analyses. But, are they consistent with a concrete system? **Can we apply the verified analyses directly to a concrete system?** For instance, can we apply the results proven in Prosa to a concrete OS?

2. In industry practice, many schedulability analyses have been implemented in commercial analyzers such as RTaW-Pegase [110] and SymTA/S [135]. However, there are no formal guarantees that the results produced by these tools are correct since: (a) as mentioned already, the underlying analyses may be flawed; but also (b) the tool implementation may contain some undetected bugs.

   In other words, there is a gap between analysis theories and their implementations. How can we provide formal guarantees for a given commercial analyzer? One straightforward solution is to formally prove the full tool implementation including the correctness of its underlying analyses. This solution has several drawbacks: For a given commercial tool,

   - the tool is usually not open-source;
   - it often uses complementary heuristic algorithms to improve its scalability, which are tricky to prove correct;
   - even when the tool is formally verified, its correctness proofs need to be updated at each software update, which may be highly nontrivial.

   It is, therefore, very time consuming and sometimes impossible to certify the full tool implementation and its subsequent versions. This suggests the alternative approach of certifying not the implementation but only its results. **Can we build formal certifiers for the results of a commercial tool?**

3. Theorem proving involves a lot of proof effort due to the fact that we cannot rely on intuitions nor omit any specifications or proof details. **How to factorize proofs to make them generic and general enough such that they can be reused for other analyses and certifiers?**

1.4 Contributions & Outline of the Thesis

Contributions

The contributions presented in this thesis address the above open problems. Each contribution is based on one of our publications [71, 59, 58].

1. We[1] implement an interface for connecting two projects involving mechanized proofs: the Prosa project for verifying schedulability analyses and the CertiKOS project for verifying OSes. This allows the two projects to benefit from each other: first, it gives a concrete instance of schedulability analysis as described in Prosa, thus ensuring that it is applicable to actual systems; second, it provides the real-time variant of the verified CertiKOS OS with formal real-time guarantees.
2. We formally verify a result certifier, named CertiCAN, which can certify the results of CAN analysis tools and can be used for industry practice. It is flexible and light-weight in the sense that it does not depend on the internal structure of the analysis tool that it complements. It is efficient enough in terms of computation time. In particular, it is able to certify results computed by RTaW-Pegase, an industrial CAN analysis tool, even for large systems. This is the main contribution of this thesis.
3. In order to factorize proofs and then to reduce the proof effort, we propose a generalized digraph task model and its corresponding analysis amenable to its formalization in Coq. The objective of this work is to formally verify the analysis for this general model such that the correctness proof of a more specific (standard or novel) analysis boils down to specifying and proving its translation into that model.

Outline of the Thesis

The thesis is organized as follows. In Chapter 2, we review real-time system models, analyses, and formalizations as well as some successful stories of result certifications. In Chapter 3, we formally prove the schedulability analysis of a real OS kernel by connecting it with the Prosa library. It permits the two projects to benefit from each other. Next, in Chapter 4, we formally prove a result certifier for verifying the results of commercial CAN analyzers. In order to formally verify many analyses and to reduce the proof effort, we propose a very expressive task model as well as its analysis amenable to its Coq formalization in Chapter 5. Finally, we conclude and present future work in Chapter 6.

[1] This is a joint work with Lionel Rieg, Maxime Lesourd, Mengqi Liu, and Zhong Shao.

Chapter 2

Real-Time Systems: Model, Analysis, and Formalization

This chapter aims at providing a general introduction to real-time system models, schedulability analysis as well as their formalization. It starts by presenting a collection of basic system models widely studied in the community, then describes several existing real-time schedulability analyses of those models. Next, we review the literature focused on increasing the confidence in real-time schedulability analyses using formal approaches. In particular, we present the Prosa library, a formally proven real-time schedulability analysis library. We discuss some success stories on result certification at the end of the chapter. These elements (system models, schedulability analyses, Prosa, result certification) form the basis of our work presented in the next three chapters.

Contents

2.1 Real-Time System Models

A real-time system consists of a set of real-time software applications executed on a platform according to a scheduling policy. A real-time system model aims at abstracting and characterizing behaviors of the system components, in order to analyze whether a considered system satisfies some properties. A proper real-time model must specify the following elements:

- **Tasks.** A task is either a piece of program dedicated to performing some functionality (*e.g.,* sampling signals or controlling actuators) or a message transmitted on a bus *e.g.,* CAN bus. A *real-time* task has timing constraints to meet *e.g.,* some critical tasks require to complete their execution within a given duration, also known as *deadline*. Most real-time tasks are recurring. We call each instance a *job*. The *response time* of a job is defined as a time duration between its arrival time and its completion time. The response time of a task is the maximum response time among all its jobs' response times.
- **Platform.** A platform represents a collection of computing and communication resources (CPUs, buses, *etc.*) as well as memories.
- **Scheduling policy.** A scheduling policy is a strategy (an algorithm) that specifies which jobs are selected to run at each time instant.
- **Time.** According to different needs or purposes, time can be modeled as *discrete*, *dense* or *continuous*. It is modeled by natural numbers, rational numbers, and real numbers, respectively. Usually, discrete time is used for expressing scheduling ticks, which suffices to express most of uniprocessor real-time theories.

In the past decades, many works have presented different models for tasks, platforms and scheduling policies. We now review the most standard ones.

2.1.1 Task models

A simple task model describes at least the frequency of arrivals and the execution time model, that is:

- How often and when does a task request to run (this is called a task arrival, or task activation)? A task can be activated once, multiple times or infinitely many times. It may arrive regularly or sporadically. For instance, in a signal processing system, the program reading sensor information will execute periodically with a fixed inter-arrival separation time (task period). In contrast, in automotive systems, the command for the brakes occurs in a non-regular way.
- How much time does a task need to complete one execution? The execution time of a task's instance (a job) is the time required to complete the job's execution without any interference. It depends on hardware performance. The execution time model of a task is defined as the worst-case (*i.e.,* longest) execution time (WCET) among all its jobs' execution times. A job represents a amount of *workload* equal to its execution time that must be serviced by the platform.

A more expressive task model is able to model task dependencies. Basically, there are two types of dependencies:

- Inter-task dependencies.
  - It can be a *timing relation* on tasks' arrivals. In the system design process, to avoid tasks arriving simultaneously, a fixed time duration may be imposed between tasks' arrivals, *e.g.,* a task always waits a fixed time duration after the arrival of another task;
  - It can be *precedence relations* on task executions. Tasks with precedence constraints should be executed according to a fixed order, *i.e.,* a task may start to execute only after the completion of another task if there is a precedence constraint between the two tasks. In multiprocessor scheduling, a program may temporarily create some parallel threads (*i.e., fork*) which execute concurrently and will then be synchronized with another thread (*i.e., join*);
  - Another type of inter-task dependencies is *resource sharing* constraints. In certain applications, some tasks may need share the same resources to execute. A task cannot execute when its shared resources are reserved by other tasks.

- Intra-task dependencies. In some cases, a task is made of a set of sub-tasks, each one with its own execution time. The execution of these sub-tasks should respect a specific order. We call this constraint is an *intra-task dependency*. For instance, in video transmissions, there are three types of frames (*i.e.,* three sub-tasks) that should be transmitted respecting a specific order. There can also be some other types of relationship between the execution time of jobs and their arrival pattern. In an automotive system, engine control commands are usually considered as tasks that arrive periodically. However, some of them are triggered by specific crankshaft rotation angles and their computational activities depend on the speed of engines. As a result, such tasks adapt their execution times based on their inter-arrival time. They are called *rate-adaptive* tasks [27].

Moreover, in order to capture more platform characteristics or to perform more precise analyses, task models can be made more expressive by adding some additional parameters such as:

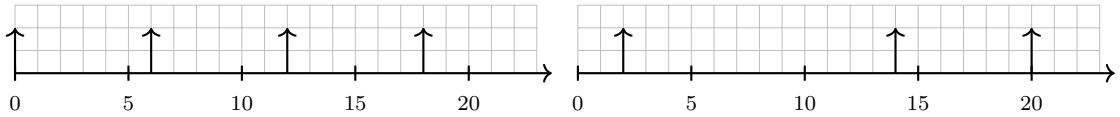
- Jitters. In control systems, tasks such as signal sampling are assumed to activate every regular time interval. However, in practice, the interval is not strictly a constant. This difference can be modeled by a parameter called *jitter* [101]. In addition, another form of jitter is the delay between a job's arrival and its detection time by a tick scheduler, *i.e.,* a task arrives at a time instant, but the scheduler does not detect it at the same time instant.
- Number of threads (for parallel computing). In parallel computing, a parallelizable task can be partitioned into several sub-tasks to be executed simultaneously on different processors. In particular, some tasks can be specified to execute on a certain number of processors/threads. To capture this feature, a new parameter, the number of threads is added, to the model.

We now present some basic task models by increasing expressivity order.

Task models considering one single WCET

The periodic task model. The first task model formally presented by Liu and Layland in 1973 [96], describes a task using two parameters (C, P). C is its WCET and P is its arrival period. A periodic task requires running every P time units and the execution time of each arrival is upper bounded by C. The task deadline D is *implicit i.e.,* $D = P$. This model is able to cover many use cases in practice such as sensory data acquisition, system monitoring, control loops, *etc.*

The sporadic task model. Mok proposed a *sporadic* task model [106] relaxing



(a) Arrivals of a periodic task (c,6) **(b)** Arrivals of a sporadic task (c,6)

Figure 2.1: Periodic arrivals and sporadic arrivals

the periodic task model. The model substitutes task periods with *minimum inter-arrival times* (or minimum separation times). A sporadic task may arrive at any time except that two successive arrivals must be separated by at least its minimum inter-arrival time. Compared to the periodic task model, the sporadic task model can model irregular signals, *e.g.,* external interrupts.

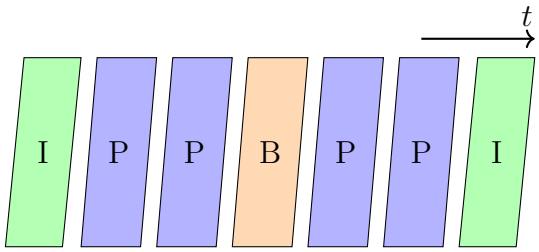
Figure 2.1 shows periodic and sporadic arrivals. Figure 2.1a illustrates the arrivals of a periodic task with a WCET c and a period 6. Figure 2.1b represents an example of arrivals of a sporadic task with the same WCET and a minimum inter-arrival time 6.

The arrival curve model. Later, Thiele et *al.* introduced the *real-time calculus* [141], whose *arrival curves* can model more arrival patterns. The arrival curve model uses a pair of functions η^+, η^- (or δ^+, δ^-) to characterize the arrival pattern of a task. For a given time duration Δ, $\eta^+(\Delta)$ and $\eta^-(\Delta)$ denote the maximum number and the minimum number of arrivals that can occur within Δ, respectively. Alternatively, for a certain number n of arrivals, $\delta^+(n)$ and $\delta^-(n)$ return the maximum and the minimum time duration needed for these n arrivals, respectively.

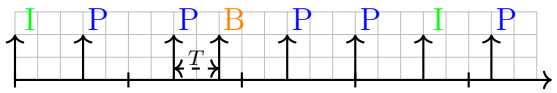
Task models considering intra-task dependencies

The above task models assume that all arrivals of a task share the same worst-case execution time. It is sufficient to model a lot of systems. However, this leads to pessimistic analysis results when such models are used with some complex applications, *e.g.,* programs with very different best and worst execution times. To tackle this issue, we present several task models that allow a task to be composed of several sub-tasks with different worst-case execution times. This introduces intra-task dependencies *i.e.,* the execution order between different types of arrivals (*i.e.,* sub-tasks). Among these models, we briefly review *the multi-frame task model* and a series of graph-based task models. Interested readers can find more details in a survey [133].

The multi-frame task model. Mok and Chen generalize WCETs of sporadic tasks by using a vector of execution times $[C^0, C^1, \ldots, C^{n-1}]$ [105]. A multi-frame task can be seen as a task with a sporadic arrival pattern as well as n WCETs. It



(a) One example of video transmission in three types of frames. The video frames are transmitted with the frame order I, P, P, B, P, P repeatedly



(b) Arrivals of the corresponding multi-frame task.

Figure 2.2: Video frames transmission modeled as a multi-frame task

arrives sporadically and its WCETs respect the order in the vector. The WCET of the $(i+1)$-th arrival is $C^{\,i \bmod n}$. Tasks deadlines are implicit. Compared to the sporadic task model, this model can model more precisely tasks, in particular those tasks have several sub-tasks and each one has it own WCET.

Figure 2.2 shows an example of a multi-frame task used in multimedia applications. A video consists of three types I, P, B of frames with their own WCETs (C_I, C_P, C_B, respectively). Video frames are transmitted respecting an order I, P, P, B, P, P repeatedly. And any two consecutive frames are separated by a minimum inter-arrival time T. This application can be modeled as a multi-frame task. One possible sequence of arrivals of this task is described in Figure 2.2b.

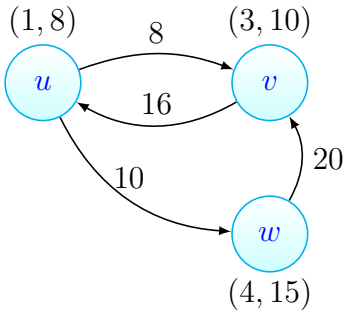
A generalized version of this model was presented by Baruah *et al.* [14]. It uses two additional vectors of n elements to model different minimum inter-arrival times $[P^0, P^1, \ldots, P^{n-1}]$ and task deadlines $[D^0, D^1, \ldots, D^{n-1}]$, respectively.

Graph-based task model. Baruah introduced the *recurring branching* task model [15], which adds branching structures to specify arrivals. Each task can be represented by a tree of job types (C, D) labeled by minimum inter-arrival times. Two other extensions use *directed acyclic graphs* instead of trees: the *recurring real-time* task model [16, 17], and the *non-cyclic recurring real-time* task model [13]. More recently, Stigge et *al.* introduced the digraph real-time (DRT) task model [129]. We focus here on the DRT task model which is expressive enough to describe all the above graph-based task models. A DRT task is specified by a graph $G_i := (V_i, E_i)$ where:

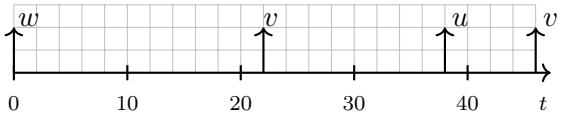
- V_i is a set of vertices representing different job types labeled by their worst-case execution time and their deadline (C, D);
- E_i is a set of edges connecting two vertices labeled with a duration $d \in \mathbb{N}$ representing the minimum inter-arrival time between jobs represented by the vertices.

The standard DRT task model assumes that task deadlines are constrained by the labels on the edges. That is, for any vertex, its deadline is less than any minimum inter-arrival time labeled on its outgoing edges.

Figure 2.3a shows a DRT task of three vertices u, v, w with their WCETs 1, 3, 4 and their constrained deadlines 8, 10, 15, respectively. A graph describes a (possibly infinite) set of paths, and each path represents a set of arrival sequences. Figure 2.3b describes one possible arrival sequence corresponding to the path $[w, v, u, v]$ of the graph in Figure 2.3a. Any two consecutive arrivals respect its minimum inter-arrival separation time, *e.g.,* instances of w and v are separated by at least 20. The execution time of each instance respects its WCET, *e.g.,* the execution time of any instance of v is less than 3.



(a) A DRT task



(b) One valid arrival sequence of the DRT task.

Figure 2.3: An arrival sequence of a DRT task

Task models considering inter-task dependencies

The DRT with extensions. Recently, the DRT model has been extended to express inter-task dependencies. Mohaqeqi *et al.* [104] proposed an extension of the DRT task model to allow the *rendezvous* mechanism, which is used for inter-task synchronizations. Abdullah *et al.* [1] used two kinds of vertices (*i.e.,* preemptive and non-preemptive) to make the DRT task model capable of taking into account resources sharing constraints between tasks.

Transaction with offsets. One of the most classic task models taking *inter-task* dependencies into account is Tindell's *offsets* model. A transaction is a collection of periodic tasks having fixed timing relations. Each task τ_i is characterized by a 4-tuple (C_i, D_i, P_i, O_i), which is used for denoting its WCET, its relative deadline, its activation period, and its *offset*, respectively. A task offset is a time duration that represents the delay between the beginning time of its transaction and its first arrival time. This model is used and analyzed in Chapter 4.

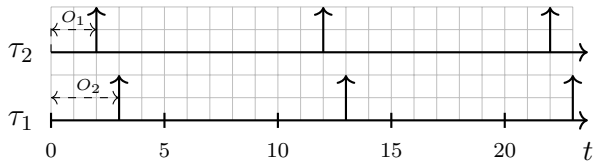


Figure 2.4: A transaction of two periodic tasks $(1, 10, 10, 2)$ and $(2, 10, 10, 3)$. The first arrivals of τ_1 and τ_2 are 3 and 2, respectively. Task τ_1 arrives always one time unit after an arrival of task τ_2.

2.1.2 Platforms

The main hardware elements constituting a platform fall into three categories:

- Processors, i.e., computing resources on which tasks execute. These can be single-core or multi-core, in which case one processor may execute several tasks at the same time. A system may consist of one or more processors.
- Memories, used to store data and programs. One distinguishes between the main memory, which is large but slow to access, and smaller but faster caches (with possibly several levels of it).
- Buses/networks. These are used to connect computation resources with other processors as well as memories. For instance, a CAN bus transmits messages between ECUs (which are made of computing elements and memories).

In the following presentation of this chapter, we focus on single-core uniprocessors.

2.1.3 Scheduling policies

Tasks complete its execution for the use of platform resources. A *scheduling policy* for a specific platform and a set of tasks is a set of execution rules that choose tasks to execute at each time instant. To make a real-time system meet its timing requirements, a scheduling policy plays a crucial role. In the past decades, many scheduling policies have been proposed. Depending on whether the scheduling decisions are made before or during system execution, scheduling policies can be classified as *off-line* or *on-line*.

2.1.3.1 Off-line scheduling policies

A scheduling policy is said off-line when all necessary scheduling decisions are computed and stored in a table before running the system. A dispatcher checks the table and sets the statically chosen tasks to execute. Systems using off-line scheduling policies have a deterministic schedule since all scheduling decisions are statically

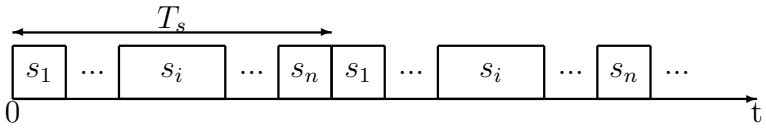


Figure 2.5: TDMA slots and cycle for a set of n tasks $\{\tau_i\}_{i=1}^{n}$ (s_i denotes task τ_i's time slot, T_s describes one TDMA cycle).

known. Off-line scheduling policies are also known as *Time Triggered* scheduling policies. One example of a single-core scheduling policy is Time-Division Multiple Access (TDMA).

The TDMA scheduling policy is defined based on the following:

1. Each task has a fixed time slot for execution;
2. Time slots are ordered in sequence to build a TDMA cycle, which repeats on the timeline. A task can be executed only during its own time slot. The length of one TDMA cycle is exactly the sum of all task time slots as shown in Figure 2.5.

Example 1. *Consider four periodic tasks $\tau_1, \tau_2, \tau_3, \tau_4$ with WCETs 2, 2, 2, 3 and periods 15, 10, 17, 14 and implicit deadlines, respectively. Assume that any instance of tasks requests its WCET to execute.*

Figure 2.6 illustrates an example of the four periodic tasks scheduled on a uniprocessor according to the TDMA scheduling policy. In the figures, ↑ represents a job's arrival time, ↓ represents a job's completion time, and the number next to ↑ denotes the job's execution time. Time slots for the four tasks are 2, 2, 2, 4, respectively. The length of a TDMA cycle is 10 time units. A task can be executed only when it is within its own time slot. If a task instance cannot finish execution within its one time slot, it must wait for the next time slot to resume execution. If a task is not active (*i.e.,* it does not have any job that has arrived but not completed yet) during its slot, the processor remains idle.

We have formally specified the TDMA scheduling policy along with its analysis for bounding the response time of tasks in Coq with the Prosa library. Interested readers can find the Coq codes here [35].

2.1.3.2 On-line scheduling policies

A scheduling policy is said on-line if scheduling decisions are made at runtime according to actual schedule states and job parameters (*e.g.,* actual execution times). A standard strategy, used in hard real-time systems, is to assign priorities to tasks

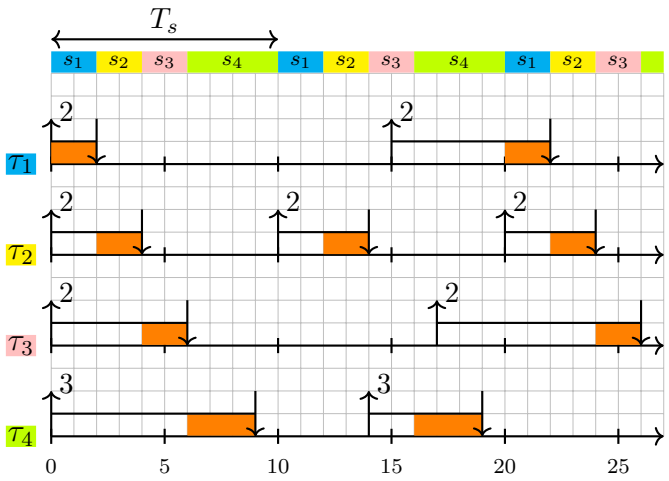


Figure 2.6: An example of the TDMA scheduling.
(The setting information is in Example 1).

according to their safety criticality levels. The scheduler always selects the task with the highest priority for execution. According to whether we use static parameters or dynamic parameters to assign priorities to tasks, scheduling policies are classified into two categories: *static priority* scheduling policies and *dynamic priority* scheduling policies.

Static priority scheduling policies. Each task is assigned a fixed priority to execute depending on its criticality level or other criteria. At runtime, the scheduler chooses among active tasks the highest priority task to execute.

Concerning hard real-time systems, all task deadlines must be met. An interesting problem is how to assign priorities to tasks so as to satisfy the overall timing constraints. Also, a good priority assignment may improve the processor utilization *i.e.,* more tasks can be executed on the processor without violating timing requirements. The following two standard priority assignment strategies are recommended for static priority scheduling policies.

Rate Monotonic (RM). The RM priority assignment policy assigns priorities to periodic tasks according to their periods: The smaller its period is, the higher its priority is.

Deadline Monotonic (DM). Similar to RM rules, the DM priority assignment policy assigns priorities to periodic tasks according to their deadlines: The smaller its deadline is, the higher its priority is.

Priority assignment has been shown important for real-time system scheduling, especially for those with a static scheduling policy. Several optimal priority assignments are proposed and proved. We refer interested readers to a review of priority assignments [44].

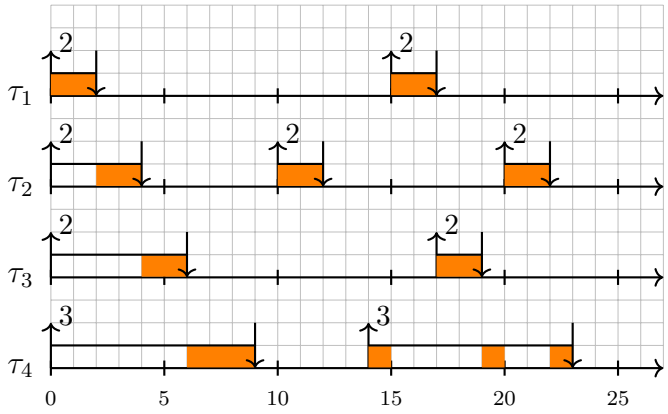


Figure 2.7: An example of the FPP Scheduling.
(The setting information is in Example 1).

We now describe the following two static priority scheduling policies, used throughout our works.

Fixed Priority Preemptive (FPP) A scheduler respects an FPP scheduling policy if the schedule is always preemptive and selects the task with the highest priority to execute among all active tasks.

Fixed Priority Non-Preemptive (FPNP). A scheduler respects an FPNP scheduling policy if and only if 1. once a task executes, it cannot be preempted until its completion; and 2. when there is no task being executed, the scheduler selects the task with the highest priority to execute among all activated tasks.

Figure 2.7 (respectively, Figure 2.8) shows the scheduling of the four periodic tasks of Example 1 using the FPP (respectively, FPNP) scheduling policy. The priority relation between the four tasks is $\tau_1 > \tau_2 > \tau_3 > \tau_4$.

In Figure 2.7, the second arrival of τ_4 starts executing at time instant 14. And then it is preempted by τ_1 (τ_3, τ_2, respectively) at 15 (17, 20, respectively), because τ_4 has the lowest priority. It is completed at 23 and its response time is 9.

In Figure 2.8, the second arrival of τ_4 starts to execute at time instant 14 and is completed at 17. At time instant 15, τ_1 with a higher priority than τ_4 arrives but it is suspended because τ_4 has started its execution and does not allow any preemption until its completion.

Dynamic priority scheduling policies. Contrary to static priority scheduling policies, in dynamic priority scheduling policies, tasks are not assigned priorities prior to execution. Task priorities are determined on-line taking advantage of run-time information. This kind of scheduling policy is more adaptive and can improve processor utilization.

Early Deadline First (EDF). A scheduler respects the EDF scheduling policy if and only if at any time instant, the job executed is the job with the earliest

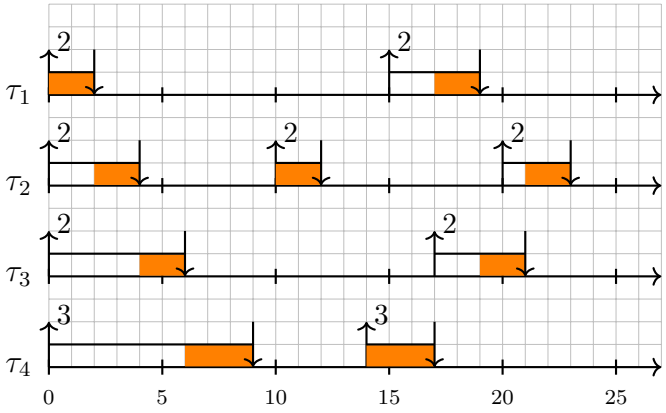


Figure 2.8: An example of the FPNP Scheduling.
(The setting information is in Example 1).

absolute deadline among all activated jobs. In other words, at any time instant, the job having the earliest deadline is assigned the highest priority to execute.

Fig. 2.9 shows an example of the same four periodic tasks as before executed under an EDF uniprocessor. The four tasks activate simultaneously at time 0 then periodically according to their periods. At time 0, the first job of task τ_2 is going to be executed because it has the earliest absolute deadline 10. At time 20, the absolute deadline of τ_3's second job is 34, while it is 30 for τ_2's third job. Hence, task τ_2 has a higher priority ($30 < 34$) than τ_3 at time 20, and τ_3 is preempted by τ_2.

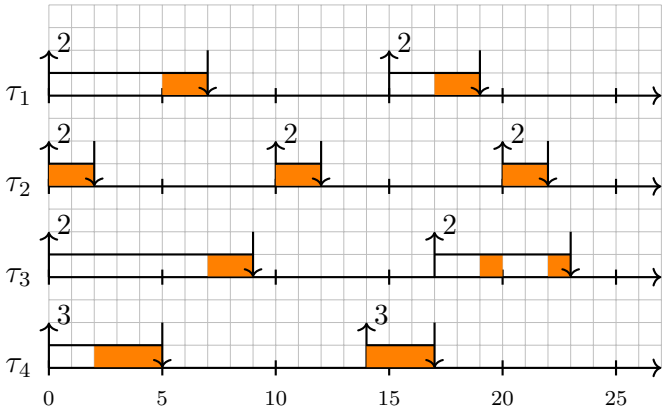


Figure 2.9: An example of the EDF Scheduling.
(The setting information is in Example 1).

Besides the above basic scheduling policies, many others have been studied. For example, Burns [26] presented the *Deferred Preemption* scheme also known *cooperative scheduling* in order to address the problem of shared resources under the mutual exclusion. To benefit from the advantage of both preemptive and non-preemptive scheduling, Wang and Saksena introduced a concept of *Preemption Thresholds* that

allows a task to disable preemption to a specified threshold priority [151]. Bertogna and Baruah [18] presented the *Limited Preemption* scheduling policy, which aims to avoid unnecessary preemptions.

2.2 Real-Time Schedulability Analyses

Real-time system analyses aim to guarantee that real-time systems satisfy their timing constraints. That is, that each task completes within its deadline. According to the criticality of effects, real-time systems can be classified into different categories:

- **Hard real-time systems.** For such systems, missing one deadline can lead to system failure. In some critical applications such as avionics, missing a deadline may have catastrophic consequences. Hard real-time systems have attracted a lot of attention in the real-time community. Many works aim at guaranteeing the absence of deadline misses.
- **Weakly-hard (or Firm) real-time systems.** Compared to hard real-time systems, weakly hard real-time systems can tolerate some deadline misses. Weakly hard real-time systems still run correctly below a certain number of deadline misses. Control systems are typical examples of weakly-hard real-time systems. In such systems, a control loop samples and adjusts command signals regularly to keep them close to a predefined target. The response time of the control loop is expected to remain within a given interval. The most important property of a control system is its stability. Usually, it is not affected when the response time of a control loop exceeds its deadline a limited number of times.
- **Soft real-time systems.** For such systems, missing deadlines does not result in system failure but in performance degradation. A soft real-time system is still expected to meet their deadlines as many times as possible. This goal is called *best effort* and is standard in *e.g.,* video transmissions.

Note that a real-time system is said hard if and only if all tasks of the system are hard, and similarly for weakly/soft real-time systems. However, in some applications like Time Sensitive Networking (TSN) [57], a real-time system may consist of a mix of different kinds of tasks (hard, weakly hard, or soft).

In this thesis, we focus on hard real-time systems. To ensure that a hard real-time system meets all task deadlines, many analyses have been proposed in the past decades. They can be classified in two categories:

- *WCET analyses.* They evaluate the WCET of a task executed on a platform.

The computation of the WCET of a task is a complex problem because it depends on the targeted system architecture as well as coordinated manipulations of system components such as I/O devices, CPU, memory, cache and so on. WCET analysis has been widely studied in the real-time community. In this thesis, we assume that the WCET of tasks are known.

- *Schedulability analyses.* They analyze when a task can finish its execution considering the scheduling policy and interference other tasks (*e.g.,* in a preemptive schedule, a task can be preempted). Schedulability analysis relies on platforms, task models, and scheduling policies as presented in Section 2.1. It is a very active field of research in the real-time community. In the rest of this section, we present basic concepts and some standards.

2.2.1 Schedulability and Feasibility

First of all, let us present some basic notions and concepts that are used in schedulability analyses.

Schedulability. Consider a platform, a set of tasks is said *schedulable* using a scheduling policy if and only if all tasks are guaranteed to meet all deadlines. Schedulability analysis aims at guaranteeing the absence of any deadline miss. A *schedulability test* is a set of conditions that guarantees that the considered system is schedulable.

Feasibility. Consider a platform, a set of tasks is said *feasible* if and only if there exists a scheduling policy which makes the set of tasks schedulable. The purpose of feasibility analyses is to find a scheduling policy under which the considered system is schedulable.

Optimality (in the sense of feasibility). Consider a platform and a set of tasks, a scheduling policy SP_o is said *optimal* if and only if: if there exists any scheduling policy SP that makes the set of tasks schedulable, the set of tasks is schedulable under SP_o.

Utilization-based test.

We now present a series of utilization-based theories related to feasibility. Let us consider a set of n periodic tasks $\Gamma := \{\tau_1, \cdots, \tau_n\}$ with implicit deadlines executed on a preemptive uniprocessor. The *utilization* of a periodic task τ is the ratio between its worst-case execution time and its period, formally, $U_\tau = C_\tau/P_\tau$. The *utilization of a system* Γ is $U_\Gamma = \sum_{1 \leq i \leq n} U_{\tau_i}$.

Static priority feasibility. Liu and Layland first presented a feasibility condition

for the FPP scheduling policy [96]. A set of n periodic tasks with implicit deadlines executed on a uniprocessor according to static priority scheduling policies is feasible when:

$$U_\Gamma \leq n(2^{1/n} - 1).$$

We know that the function $n \mapsto n(2^{1/n} - 1)$ is monotonically decreasing and the limit of $n(2^{1/n} - 1)$ as n approaches infinity equals $ln(2)$. Therefore, the system is feasible when its utilization is less than $0.69 \approx ln(2)$. Note that this is a sufficient but not necessary condition: it is sometimes pessimistic. A tighter condition can be computed when the number of tasks in the considered system is known. For instance, when $n = 2$, the system is feasible when the utilization is less than $0.83 \approx 2(2^{1/2}-1)$.

Liu and Layland showed this feasibility condition using the FPP scheduling policy based on the RM priority assignment algorithm (we call it *the RM scheduling policy* for short in the following[1]). They proved that the RM scheduling policy is optimal when task deadlines are implicit. Later, Leung and Whitehead [93] proved that the DM scheduling policy is optimal when task deadlines are constrained by task periods $D \leq P$. When deadlines are arbitrary, neither the RM nor the DM scheduling policies are optimal [90].

Dynamic priority feasibility. Liu and Layland [96] also showed that a set of periodic tasks with implicit deadlines scheduled on a uniprocessor is feasible if and only if:

$$U_\Gamma \leq 1.$$

They also proved that the EDF scheduling policy can deal with such utilization and is optimal. Later, Dertouzos showed that the EDF scheduling policy is optimal even for arbitrary deadlines [47].

Utilization-based tests are used to check the feasibility of a set of tasks. An alternative approach is based on response time analysis (RTA). We present some basic RTAs in the following subsection.

2.2.2 Response Time Analysis

The response time of a job is defined as the duration between its arrival time and its completion time. RTA is used for computing the worst-case response time (WCRT) of all jobs of a considered task, *i.e.,* the task's WCRT. Using RTA, we can perform a tighter or even exact schedulability test by just comparing a task's WCRT to its deadline.

This section presents some seminal results concerning RTAs. All results apply

[1]Similarly for the DM scheduling policy.

to a system made of a set of n periodic/sporadic tasks $\Gamma := \{\tau_1, \cdots, \tau_n\}$ executed on a uniprocessor according to the FPP or FPNP scheduling policy. Each task τ_i is characterized by its WCET C_i and its period/minimum inter-arrival time T_i and a deadline D_i. It is assigned a unique (that is, two tasks cannot have the same priority) priority noted by its subscript i.

First, let us introduce some basic notions needed to present RTAs. We denote $hep(i)$, $hp(i)$, $lp(i)$ the sets of tasks of the system under study whose priorities are higher than or equal to, higher than and less than i, respectively.

Task workload bound. During analyses, tasks are usually abstracted by *workloads* that are defined as follows. Consider a task τ and a time duration Δ, the workload of this task within this duration is the maximum cumulative execution time requested by τ. For example, for a periodic/sporadic task τ_i, its workload is bounded by $\lceil \frac{\Delta}{P_i} \rceil C_i$.

Critical instant. A critical instant of a task is the worst-case scheduling scenario for that task, such that its WCRT is reached. Liu and Layland [96] proved that for any task τ_i, its critical instant occurs when all tasks with a priority higher than or equal to i (*i.e.,* $hep(i)$) activate simultaneously.

2.2.2.1 RTA with constrained deadlines

RTA for the FPP scheduling policy

Joseph and Pandya [78] provided a seminal result for a set of periodic/sporadic tasks with constrained deadlines using the FPP scheduling policy. It is based on the fact that the critical instant occurs when all tasks activate simultaneously. The WCRT of task τ_i is the least fixed point of the following equation.

$$R_i = C_i + \sum_{\tau_j \in hp(i)} \lceil \frac{R_i}{P_j} \rceil C_j \tag{2.1}$$

As shown in this formula and illustrated in Figure 2.7, the response time R_4 of task τ_4 is equal to the sum of τ_4's WCET (that is 3) and all workload requested (that is 2+2+2=6) by $hp(i)$ within a duration equal to R_4 (that is 9). R_i is computed by iterations which are guaranteed to terminate if the system utilization is less than 1. Note that the computed result R_i is the WCRT of task τ_i only if $R_i \leq P_i$, otherwise, it may not be correct as two jobs of τ_i may interfere with each other, which is not taken into account in Equation 2.1.

Refuted RTA for the FPNP scheduling policy

Later, Tindell *et al.* [147] proposed a RTA for the FPNP scheduling policy in the context of analyzing CAN buses. A task executed on a non-preemptive schedule cannot be preempted by other tasks even if they have a higher priority. This has two consequences:

1. *Queueing delay.* For any job, its *queueing delay* [2] is the duration between its activation and the beginning of its execution. Given that a job will continue to execute until its completion once it starts its execution, its response time is the sum of its queueing delay and its execution time.

2. *Blocking factor.* During a queueing delay, a task may be blocked by a task with a lower priority. We call this blocking time the *blocking factor*, which is a part of the queueing delay. According to the FPNP scheduling policy, a task can be blocked at most once by a lower priority task.

Following the same intuition as for the FPP case, the worst-case queueing delay Q_i of a task τ_i is defined as the least fixed point of the following equation:

$$Q_i = B_i + \sum_{\tau_j \in hp(i)} \left\lceil \frac{Q_i + \epsilon}{P_j} \right\rceil C_j \tag{2.2}$$

Where $B_i = \max_{\tau_h \in lp(i)}(C_h - \epsilon)$ is an upper bound of the blocking factor for task τ_i. The positive constant ϵ is very small and used to account for the fact that 1. all jobs from $hp(i)$ must be completed strictly before the end of the queueing delay Q_i; and 2. there is no blocking time for the task with the lowest priority. For discrete time models, ϵ is 1.

As a result, one concludes that $WCRT_i$ under the FPNP scheduling policy is the sum of its queueing delay and its WCET,

$$WCRT_i = Q_i + C_i \tag{2.3}$$

Actually, the Equation 2.2 used for computing Q_i is incorrect (too optimistic). The flaw was reported by Bril *et al.* [25] and corrected by Davis *et al.* [43] more than one decade after the original paper. The issue is that this analysis assumes that the WCRT of τ_i can be found within one period of τ_i after the critical instant. It is the case for the FPP scheduling policy but not for the FPNP scheduling policy. A counterexample is presented in Figure 2.10. The figure shows three tasks τ_1, τ_2, τ_3 with implicit deadlines and the same WCET 4 and periods 10, 14, 14, respectively. Their priority relation is $\tau_1 > \tau_2 > \tau_3$. We now focus on computing $WCRT_3$. As

[2] Sometimes, we also call it *queuing prefix* that is formally defined in Chapter 4 and Chapter 5.

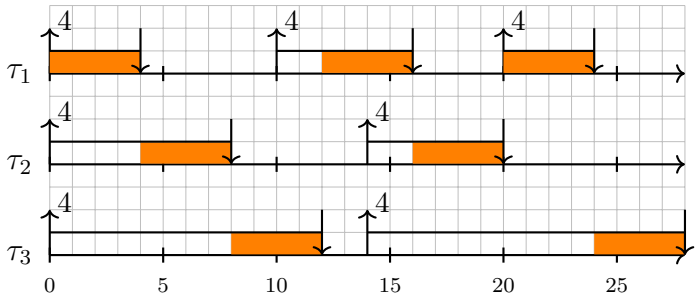


Figure 2.10: A counterexample for the refuted RTA.

a first step, let us calculate Q_3 using the formula 2.2, where $B_3 = 0$ because τ_3 is the task with the lowest priority. We set 0 as Q_3's initial value denoted Q_3^0 then get $Q_3^1 = \lceil \frac{Q_3^0+1}{10} \rceil \times 4 + \lceil \frac{Q_3^0+1}{14} \rceil \times 4 = 8$ ($Q_3^2 = 8$, respectively) by the first (the second, respectively) iteration. As it converges ($Q_3^1 = Q_3^2$) after two iterations, $Q_3 = 8$. Therefore, the WCRT of τ_3 is $8 + 4 = 12$ using formula 2.2. However, it is incorrect due to the fact that the response time of the second instance of task τ_3 is 14 which is greater than 12. A corrected RTA for the FPNP scheduling policy is presented in the next sub-section.

The FPNP scheduling policy is widely implemented, *e.g.,* in the CAN communication protocol. Millions of CAN buses/ECUs are integrated into cars or industrial vehicles every year. Any bug or flaw is unacceptable in such safety-critical domain. This flaw can be seen as a motivation to build formal proofs for real-time system analyses.

2.2.2.2 RTA with arbitrary deadlines

RTA for the FPP scheduling policy

Let us first focus on the FPP scheduling policy. Formula 2.1 is not sufficient when task deadlines are not constrained, *i.e.,* $D > P$. Lehoczky [90] developed an exact schedulability criterion for $D = mP$ (where m is a positive integer) and Tindell *et al.* presented a general RTA for arbitrary deadlines. Both results are based on the new *busy window* concept that we now describe. The idea is to compute $WCRT_i$ by examining together all jobs of τ_i that may interfere with each other.

Busy window (or busy period [90, 64]). A *level-i busy window* is a time interval $[t_1, t_2)$ within which

- the processor is continuously occupied by jobs with a priority higher than or equal to i, and
- there does not exist any job with a priority higher than or equal to i that

arrived strictly before t_1 (t_2, respectively) but not be completed at t_1 (t_2, respectively).

Lehoczky proved that the WCRT of τ_i can be found in the level-i busy window that starts with a critical instant (*i.e.,* at which all tasks with a priority higher than or equal to i activate). Consider the q-th ($q = 1, 2, \ldots$) job of task τ_i activated in that busy window, the duration between the start of the busy window and the job's completion can be specified by:

$$w_i(q) = qC_i + \sum_{\tau_j \in hp(i)} \left\lceil \frac{w_i(q)}{P_j} \right\rceil C_j \tag{2.4}$$

$w_i(q)$ corresponds to q times the execution time of τ_i and all higher priority workload that may be requested during that time duration, it can be found by computing the least fixed point of Equation 2.4. The formula converges when the system utilization is less than 1. The response time of the q-th job of task τ_i in the window is $R_i(q) = w_i(q) - (q-1)P_i$ since that job arrives $(q-1)P_i$ after the first job of τ_i is aligned with the beginning of the busy window. Finally, $WCRT_i$ is the largest one among the response times of jobs of task τ_i activated in the busy window. Formally,

$$WCRT_i = \max_{q=1,2,\ldots,q_i^+} R_i(q) \tag{2.5}$$

where q_i^+ is the maximum number of jobs to be examined, it can be computed by $\lceil \frac{L_i}{P_i} \rceil$, where L_i is defined as the least fixed point of the following equation:

$$L_i = \sum_{\tau_j \in hep(i)} \left\lceil \frac{L_i}{P_j} \right\rceil C_j \tag{2.6}$$

A faster way for computing q_i^+ is to find the smallest number q such that the condition $w_i(q) \leq qP_i$ holds.

RTA for the FPNP scheduling policy

As mentioned earlier, the original RTA for the FPNP scheduling policy was corrected by Davis *et al.* [43] based on George *et al.* [62]'s analysis. We now present the corrected version. Similar to the arbitrary deadline case for the FPP scheduling, the analysis relies on the busy window concept. It examines all jobs activated in a busy window (even for constrained deadlines) to compute $WCRT_i$. The queueing

delay[3] of the q-th job of task τ_i in the window is the least fixed point of the following equation.

$$Q_i(q) = B_i + (q-1)C_i + \sum_{\tau_j \in hp(i)} \left\lceil \frac{Q_i(q) + \epsilon}{P_j} \right\rceil C_j \tag{2.7}$$

This is a direct generalization of Equation 2.2. The difference with the faulty analysis lies in the stopping condition, *i.e.,* how many jobs of τ_i must be analyzed, as explained below. The response time of the job is

$$R_i(q) = Q_i(q) - (q-1)P_i + C_i \tag{2.8}$$

Therefore, the $WCRT_i$ under the FPNP scheduling policy is

$$WCRT_i = \max_{q=1,2,\ldots,q_i^+} R_i(q) \tag{2.9}$$

where q_i^+, the maximum number of jobs of task τ_i to be checked, can be determined by using an upper bound L_i of the length of the busy window. That is $q_i^+ = \lceil \frac{L_i}{P_i} \rceil$ with L_i being the least fixed point of the following equation:

$$L_i = B_i + \sum_{\tau_j \in hep(i)} \left\lceil \frac{L_i}{P_j} \right\rceil C_j \tag{2.10}$$

Note that a level-i busy window does not necessarily close if the q-th job of τ_i completes before $q+1$-th job arrives. This would, *e.g.,* be the case for the counterexample of Figure 2.10, which fixes the problem. All the above analyses are performed considering hard real-time systems running on uniprocessors. They are useful to understand basic real-time theories as well as the research described in this thesis. In the literature, there exist, however, quite a few different analysis approaches for analyzing different kinds of systems. We discuss some of them in the following subsection.

2.2.3 Related analysis approaches

Typical worst-case analysis. Weakly hard real-time systems can tolerate a bounded number of deadline misses without jeopardizing the system behavior. Even though we can apply the analyses for hard real-time systems directly to weakly hard real-time systems, it is constraining. Typical Worst-Case Analysis (TWCA) [114, 157] is dedicated to analyzing weakly hard real-time systems and

[3]Note that the queueing delay of a job is generalized as the duration between the beginning of its corresponding busy window and the beginning of its execution.

aims at providing a bounded number of potential deadline misses. This approach has been shown suitable for analyzing closed-loop system models and verifying its corresponding properties using Logical Execution Times (LET) techniques [61].

Stochastic analysis. Díaz *et al.* [49] present *Stochastic analysis* to compute probabilistic response time distributions by relaxing the assumption on the WCET of jobs. It was later improved by several works [77, 102]. One motivation of this analysis is that in many engineering applications, a small failure rate of systems is acceptable. As specified in the automotive safety standard ISO26262, the safety level could be quantified by a failure rate: 10^{-9} per hour for ASIL D [4], 10^{-8} for ASIL C and B, and 10^{-7} for ASIL A [45]. Also, for soft real-time systems, missing deadlines does not lead to system failure but can degrade the quality of services provided by the systems.

Compositional performance analysis. Compositional performance analysis (CPA) [65, 120] is an approach for performance analysis of complex heterogeneous embedded systems *e.g.,* distributed communication systems. In such complex systems, there are many subsystems interacting with each other. To find end-to-end latencies (*i.e.,* WCRTs for task chains) in a distributed real-time system, Tindell [146] presented a holistic analysis by considering both processor scheduling and communication delays. However, in practice, the holistic analysis cannot deal with modern complex distributed real-time systems due to its computation complexity. The main idea of CPA is to allow subsystem integrations without changing local RTAs. It models each subsystem using event streams $E(t)$, which is a function that returns the accumulated number of events activated up to the time instant t. Then it iteratively determines end-to-end latencies by considering the propagations of event streams at the system-level. It was first developed by Gresser [65], and refined by Richter and Schliecker [118, 122].

Network calculus. Very similar to the CPA theory but not relying on RTAs, the network calculus theory [39, 34, 89] was independently developed for guaranteed quality of service networks through bounding network latencies. It characterizes tasks/services traffic arrivals by a cumulative function $A(t)$, which counts the maximum accumulated workload arrived until time instant t. Network calculus is based on the Min-Plus algebra and computes network delays. It was first presented by Cruz [39] and then completed and improved by many [40, 121, 3, 41, 33, 88, 2]. Interestingly, Thiele *et al.* [141] adapt network calculus notions to real-time scheduling. This network calculus perspective provides new insights into real-time system

[4] ASIL stands for Automotive Safety Integrity Level with ASIL A the lowest level and ASIL D representing the highest level.

models and analyses.

A recent effort aims at combining different models or analysis approaches to improve the precision of analyses. Boyer and Roux [24] proposed a common model for the flow model of network calculus and the event stream model of CPA. Based on this work, Köhler *et al.* [82] proposed a more accurate CPA model by introducing a new component interface. In parallel, Boyer and Doose [23] presented a novel technique by combining network calculus and RTAs for improving the analysis of network delay bounds. Daigmorte and Boyer [42] improved RTAs for CAN bus protocol by using network calculus. All these works would benefit from comparisons and combinations of the different approaches where applicable.

2.3 Formal Methods for Real-Time Schedulability Analyses

The complexity of real-time systems and analyses make them error-prone. A reason is that system models are usually described using informal languages. It is difficult to verify their consistency. Another reason is that most published results are based on *pen-and-paper* proofs which often rely on intuitions and omit many details. This has motivated the use of *formal methods* to provide formal guarantees to real-time theories.

Formal methods are mathematical techniques that use formal languages for specifying and solving problems. Both academic and industry advocate to use formal methods to increase the safety or security levels of research results and industrial products. In international standards such as ISO26262 or Common Criteria, formal verification achieves the highest safety and security level. Formal methods theories and tools are now mature and have been successfully applied to system (hardware and software) verification. Importantly, formal methods not only provide mathematically rigorous guarantees to the correctness of analyses results but also permit a much better understanding of models and analyses.

Formal verification for real-time system analysis has been investigated for more than two decades. In this section, we mainly review previous work on real-time system analysis using formal approaches, especially those using *model checking* and *theorem proving*.

2.3.1 Model checking of real-time systems

Model checking is a formal verification technique based on finite-state automata to model systems. It is able to automatically check whether a model satisfies properties. Thanks to its automatic verification procedure, model checking has been successfully applied to the verification of hardware and software. However, finite-state automata have no timing aspects. In order to verify the timing behavior of real-time systems, model checking has been extended to *Timed automata* [5], *Stopwatch automata* [4], and *Hybrid automata* [4, 74].

Using Stopwatch automata [4], Corbett [36] proposed a formal model for modeling scheduling algorithms with real-time tasks as well as some features such as resource sharing, priority preemption, and task suspension. However, the reachability analysis of Stopwatch automata is undecidable in general [36].

Fersman *et al.* [55] extended timed automata with asynchronous tasks to model schedulability analyses. Based on this model, they proved that the schedulability checking problem is decidable. Later, Fersman *et al.* [54] proposed a model (a timed automation with two extra clocks[5]) to model and to analyze real-time scheduling properties for uniprocessors respecting fixed priority scheduling policies. This approach has been implemented in Times [6], a tool to check schedulability analyses for uniprocessors. Guan *et al.* [70] extended this approach to model schedulability analyses for periodic tasks on multiprocessors. Cordovilla *et al.* [37] proposed a multiprocessor schedulability analyzer by combining a search technique with the UPPAAL model checker [87]. This analyzer is capable of analyzing periodic tasks executed on multiprocessors according to static or dynamic scheduling policies. Baker and Cirinei [11] described a necessary and sufficient condition for schedulability of sporadic tasks executed on multiprocessors according to the EDF scheduling policy. This work relaxed the assumption on periodic tasks in early works. Using the UPPAAL tool, Kim *et al.* [79] showed how to formally validate and analyze system design models at an early stage by combining statistical and symbolic model checking.

More recently, Sun and Lipari [134] applied linear hybrid automata to model an exact schedulability analysis for a set of sporadic tasks executed on a multiprocessor respecting a global FPP scheduling policy. The authors used pre-order simulation relations to avoid the state-space explosion problem and introduced a concept of decidability interval to model timing constraints. Huang *et al.* [75] used hybrid automata to analyze weakly-hard systems to guarantee that tasks cannot miss up to a bounded number of deadlines for a certain number of arrivals. They also

[5]In addition to the original model.

discussed relaxation techniques and over-approximation approaches.

Different from the above techniques, Lime and Roux [95] used an extension of Petri Nets for the verification of timed properties for real-time systems with a preemptive scheduling policy. In particular, they show how to deal with fixed priority and EDF scheduling policies.

The main strengths of model checking are:

- It is a general formal verification technique that can be applied to both academic researches and industrial applications. Based on sound mathematical supports such as graph theories and logic, it can provide high confidence on checked results;
- It is a completely automatic process. Thanks to this feature, model checking has been widely and successfully used for verifying behaviors of hardware and software;
- It is able to provide a counterexample when a property is violated. This information is very useful for debugging in the design process.

The main limitations of model checking are:

- The biggest issue is that model checking suffers from a state-space explosion problem. In schedulability analyses, there is a scalability problem using model checking [134] and it cannot be applied to verify the schedulability of relatively large systems.
- The verified results are only as good as the abstract model is. If the actual model is not equal to the abstract one, we have to ensure that it is a subset of the abstract model (*i.e.,* the abstract model is an over-approximation of the actual model) in order to use the verified results.

2.3.2 Theorem proving real-time system analyses

Theorem proving is another formal verification approach which aims at building formal proofs for mathematical theorems using computer programs. Several interactive theorem provers such as Coq [138], PVS [140], Isabelle/HOL [139] have been developed in order to help build large-scale mechanized proofs. They have been successfully used to formally verify OSes such as CertiKOS [66], compilers such as CompCert [91], mathematical theorems such as the Feil-Thompson theorem [63] and so on. We now present some mechanized proofs of scheduling analyses.

Mechanized proofs for real-time schedulers/kernels.

Fidge *et al.* [56] formalized a simple round-robin real-time scheduler for the

MIPS R3000 RISC processor and proved its correctness using the Ergo theorem prover [53, 149]. Later, Wilding *et al.* [153] specified and analyzed a cyclic scheduler using the PVS theorem prover. Tol [148] formally verified a simple version of a real-time kernel using Nqthm (also known as the Boyer-Moore theorem prover). In this kernel, tasks are scheduled according to the EDF scheduling policy. Recently, Xu *et al.* [155] provided a verification framework for real-time OS kernels using the Coq proof assistant. They have applied this framework to verify the main parts of the commercial μC/OS-II real-time kernel [103], and proved a simple scheduling property (the executing task has the highest priority). Similarly, Andronick *et al.* [7] proposed a framework for formal verification of real-time kernels using the Isabelle/HOL proof assistant. Using this framework, the authors proved the same simple scheduling property for the commercial eChronos real-time operating system (RTOS) [52]. More recently, Liu *et al.* [99] presented a compositional framework for real-time preemptive kernels by extending a verified single-core OS kernel, *i.e.,* mCertiKOS [66, 38]. This work uses Coq and *virtual timelines* (*i.e.,* supply and demand functions specified in [97]) for proving schedulability properties.

Mechanized proofs for real-time scheduling analyses.

Wilding [152] first established a machine-checkable proof of EDF optimality for periodic tasks executed on uniprocessors using the Nqthm theorem prover. The author pointed out that formal proofs require a lot of efforts because of the need to formalize all details. For instance, a lot of concepts needed by formal proofs are not explicit in the original informal proofs. We refer interested readers to Wilding's dissertation [154] for a more detailed comparison of formal and informal proofs. Zhang *et al.* [161] formally verified EDF optimality on uniprocessors using the Propositional Projection Temporal Logic (PPTL) system.

Sinha and Suri [126] formalized and verified the RM optimality using the PVS proof assistant. They demonstrated the capacity of a proof assistant to formalize a fault-tolerant real-time protocol, and identifying flaws in existing works when formally specifying them. Later, Varma [150] proposed a modular formal analysis of fault-tolerant real-time allocation and scheduling policies. The author showed that the proof effort could be reduced by reusing previously proven theorems.

Dutertre [50] developed a formal specification of the *priority ceiling protocol* [124] using PVS. In online scheduling, a task may be blocked by another task with a lower priority when considering shared resources. This phenomenon is called *priority inversion*. The priority ceiling protocol is designed to bound blocking times caused by priority inversion and increasing scheduling predictability. The author provided formal proofs in PVS for the schedulability analysis of a set of periodic tasks dis-

patched on a uniprocessor according to the FPP scheduling policy and the priority ceiling protocol. Dutertre and Stavridou [51] discussed extensions of this work and highlighted the benefits of formal specifications and proofs in real-time scheduling problems.

The original informal specification of the *priority inheritance protocol* [124] was flawed [158]. This motivated Zhang *et al.* to build a formal precise specification for the protocol and also to provide a rigorous proof for its correctness [162]. Following this work, Zhang *et al.* [163] proved a finite bound on priority inversions using Isabelle/HOL, which improved their early result. Also, they provided a brief survey on existing works concerning this protocol and pointed out a list of formal publications [94, 86, 97, 115, 125, 124] specifying incorrect behaviors. Zhang *et al.* 's work showed that building mechanized proofs not only provides high confidence in theories but also permits to have a better understanding of concept specifications.

De Rauglaudre [46] was the first to use the Coq proof assistant to formalize and prove real-time scheduling problems. Specifically, the author proved a schedulability criterion (known as Jan Korst's theorem presented in [83, 84]) for a set of periodic tasks executed on a uniprocessor according to the FPNP scheduling policy.

Recently, Cerqueira and Brandenburg [29] launched the Prosa project [112], which aims at building a proven schedulability analysis library using the Coq proof assistant. As the first case study, Cerqueira and Brandenburg [29] formalized global RTAs [19] for both the FP and EDF scheduling policies on multiprocessors as well as two extensions for released jitters and parallel jobs. Later, a lot of other schedulability analyses including the main results of this thesis have been formalized in Prosa. To the best of our knowledge, the Prosa library has been recognized as the largest framework of formally proven real-time schedulability theories. In the next section, we will give a detailed presentation of the Prosa library through a simple formalization.

The main strengths of theorem proving are:

- It provides high confidence in the correctness of a theory. Modern theorem proving systems themselves such as Coq have attained the utmost level of reliability (*i.e.,* respecting the de Bruijn criterion and/or the LCF principle). Also, such tools use formal languages and mathematical logic to specify and prove theorems, which requires to make all proof details explicit. And the proofs are mechanically checked.

- Some proof assistants support the extraction technique, which permits to obtain certified programs. Using such proof assistants, we can implement an algorithm and prove some of its properties and then, from the proven algo-

rithm, extract a compilable program which satisfies the proven properties. This feature is able to fill the gap between theory and practice as it can ensure that a program is correct in both its theory and its implementation (Note that it relies on the trusted code base, including the code generation mechanism, the compiler *etc.*.);

- Compared with model checking, it can be used for proving arbitrarily complex problems, and it does not suffer from the state-space explosion problem. For example, the *Feit-Thompson theorem* [63] has been formally verified using Coq.
- Moreover, a proof assistant makes proving theorems an interactive process. It is fun to build proof scripts. As said by Leroy [92], "Building such scripts is surprisingly addictive, in a videogame kind of way...".

The main limitations of theorem proving are:

- It is time-consuming. In contrast to *pen-and-paper* proofs, mechanized proofs do not admit any omission in specifications and proofs. In contrast to model checking, it does not provide much automation.
- It needs experts to formalize specifications, theorem statements, and conduct proofs. Using a proof assistant, the proof itself can be mechanically verified, but all specifications and statements are still required to be validated by experts.

As the developments described in this thesis are carried out within the framework of Prosa, we now present Prosa though a simple formalization.

2.4 Formalization in Prosa

The Prosa library[6][112, 31] was initially developed by Cerqueira and Brandenburg for formally proving existing and new schedulability analyses using the Coq proof assistant. The authors prioritize readability to make the project accessible to researchers familiar with real-time scheduling theories but without prior experience in formal methods. This feature makes the formal specifications in Prosa easily reviewed, understood and validated by the community.

The library is now organized into four basic components as shown in Figure 2.11, and also contains an additional component *Results* for collecting high-level results.

[6]It is (being) further developed through a French project CASERM [117] and a French-German RT-Proofs project [137].

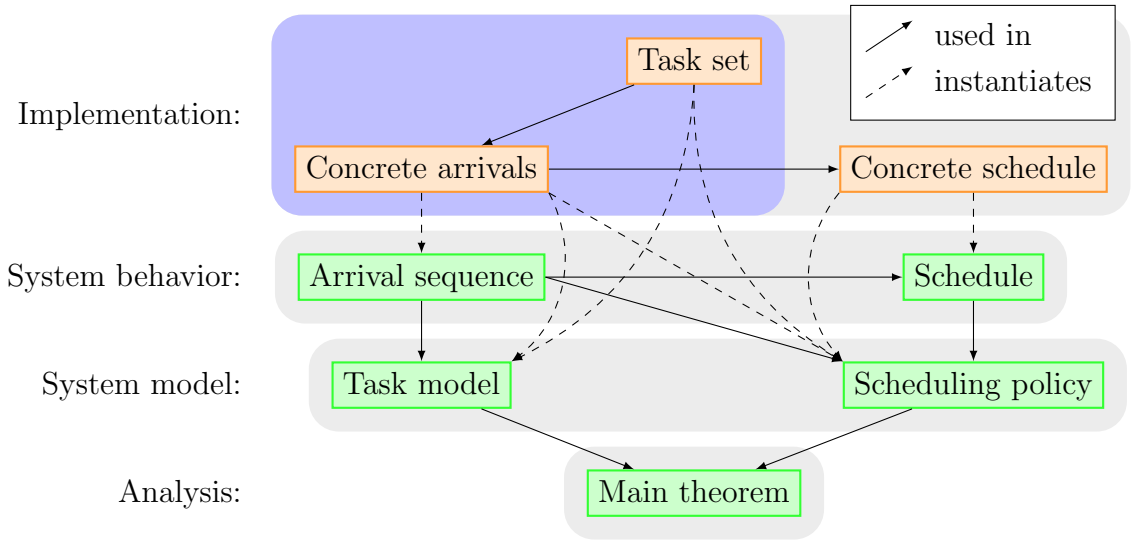


Figure 2.11: An overview of Prosa layers

System behavior The base representation of system behavior is based on discrete time traces representing infinite sequences of events. There are two such kinds of sequences: arrival sequences which record requests for service called job activations and schedules which record processor states, *e.g.,* scheduled jobs, overhead. For instance, for an ideal uniprocessor (in which there is no overhead), at any time instant, a processor state is that there is a job scheduled or the processor is idle.

System model Based on system behavior, task models (arrival patterns and execution time models), processors, scheduling policies can be defined. A system model defines a set of possible behaviors that satisfy it. These models and policies are axiomatic in the sense that they are given as predicates on arrival sequences/schedules, not as generating and scheduling functions. In this component, some properties of these models are provided and proved.

Analysis This part of the library contains definitions needed for various analyses (*e.g.,* busy window, schedulability) as well as the actual proofs of response time and schedulability analyses.

Implementation This library component contains implemented examples of arrival sequences and schedulers. It aims at proving the consistency of the considered system models.

Results An additional component is used for collecting proven high-level analysis results. It actually contains results of EDF optimality and RTAs for EDF, FPP, FPNP and so on.

We now present the Prosa library in more detail using a simple example: a schedulability analysis of a set of sporadic tasks executed on a uniprocessor according to the FPP scheduling policy.

2.4.1 System Behavior

The basic definitions in Prosa relate to concrete system behavior. Time is discrete and can be seen as scheduling ticks: durations are given in numbers of ticks and instants are given as numbers of ticks from the initialization. As key elements in both arrival sequences and schedules, *jobs* (*i.e.,* instances of tasks) are characterized as follows.

Definition 1 (Job)**.** *A job $\jmath$ is an instance of a* task $\tau(\jmath)$ *with a positive cost (Prosa uses this terminology for execution time) denoted* $\mathbf{c}(\jmath)$*.*

In some cases, a job is associated with a unique identifier. We do not use the identifier directly, it is only used to distinguish jobs of the same task in traces. An *arrival sequence* is a trace of job activations, from which the actual workload that must be scheduled at a given time instant can be deduced.

Definition 2 (Arrival sequence)**.** *An* arrival sequence *is a function ρ mapping any time instant t to a finite (possibly empty) set of jobs $\rho(t)$. A given job appears at most once in an arrival sequence.*

The *arrival time* of a job $\jmath$ appearing in an arrival sequence is given by the function $\mathbf{a}(\jmath)$. The fact that the job $\jmath$ appears at instant t in arrival sequence ρ is formalized as $\jmath \in \rho(t)$, *i.e.,* $\jmath \in \rho(t) \Rightarrow \mathbf{a}(\jmath) = t$.

The scheduler is not modeled as a function, instead, we work with *schedules* which are traces of processor states reflecting schedule information (*e.g.,* scheduled jobs, cores on which jobs are scheduled, overheads *etc.*).

Definition 3 (Schedule)**.** *A* schedule *is a function σ mapping any time instant t to a processor state.*

This definition is generic, and we obtain a schedule for a specific platform by specifying the processor states. For instance, a schedule for a so-called ideal uniprocessor is defined as follows:

Definition 4 (Uniprocessor schedule)**.** *A* uniprocessor schedule *is a function σ mapping any time instant t to either the job scheduled at time t or $\bot$.*

This reflects the fact that for uniprocessors, the processor is either scheduling a job or idle (in other words, a uniprocessor cannot schedule more that one job at a time). Given an arrival sequence ρ and a schedule σ over ρ, a job $\jmath \in \rho$ is said to be *scheduled at an instant* t if $\sigma(t) = \jmath$, the *service* received by $\jmath$ up to time t is the number of instants before t at which $\jmath$ is scheduled (this reflects the fact that all scheduling overheads are assumed to be zero). A job $\jmath$ is said to be completed at time t if the service it received up to time t is equal to its cost $\mathbf{c}(\jmath)$. A job $\jmath$ is said to be *pending* at time t if it has arrived before time t and is not completed at time t. In a well-formed schedule, only pending jobs can be scheduled. From now on, we only consider well-formed schedules. A job $\jmath$ is said to be schedulable if it is completed before its *absolute deadline*. The absolute deadline of a job is defined by its arrival time and its task's relative deadline $\mathbf{d}(\jmath) := \mathbf{a}(\jmath) + \mathbf{D}(\tau(\jmath))$ where $\mathbf{D}(\tau(\jmath))$ represents the corresponding task deadline.

2.4.2 System Model

To specify system models abstracting from their possible behaviors, Prosa defines task models and scheduling policies based on a set of predicates on traces *i.e.,* arrival sequences and schedules. Response time analysis of such models can then provide guarantees on all behaviors satisfying the analyzed model.

To illustrate this kind of definitions and specifications, we focus on the *sporadic* task model and the FPP scheduling policy.

Specification of the sporadic task model

Definition 5 (Task)**.** *A task* τ_k *is defined by a deadline* $\mathbf{D}(\tau_k) \in \mathbb{N}^+$ *and a WCET* $\mathbf{C}(\tau_k) \in \mathbb{N}^+$.

A sporadic task is characterized by an additional parameter: its minimal inter-arrival time $\mathbf{P}(\tau_k) \in \mathbb{N}^+$. Its deadline is implicit *i.e.,* $\mathbf{D}(\tau_k)$ is equal to $\mathbf{P}(\tau_k)$. Task priorities are defined by a relation order in Prosa. For the sake of simplicity, we specify here a task's priority using its subscript *i.e.,* the priority of task τ_k is $k \in \mathbb{N}$. The smaller the number is, the higher priority is.

The sporadic task model (presented in Section 2.1) is specified by a sporadic arrival pattern and an execution time model.

The sporadic arrival pattern is a constraint on job arrival times: consecutive arrivals of a task τ are separated by a minimum inter-arrival $\mathbf{P}(\tau)$. An arrival sequence ρ is sporadic if for any two distinct jobs $\jmath_1$, $\jmath_2 \in \rho$[7] of the same task τ,

[7]Here, we omit the time t from $\rho(t)$ for short and the complete notation is $\jmath_1 \in \rho(t_1)$, $\jmath_2 \in \rho(t_2)$.

$|\mathbf{a}(j_1) - \mathbf{a}(j_2)| \geq \mathbf{P}(\tau)$. Periodic arrivals are a particular case of this model where $\mathbf{P}(\tau)$ is the task period. We denote $Sporadic(\rho, S)$ a predicate on an arrival sequence ρ and the corresponding system S to check whether this arrival sequence satisfies the sporadic model. Note that the response time analysis which calculates the WCRT of tasks scheduled according to an FPP policy on uniprocessors provides the same bounds for sporadic and periodic task models.

The execution time model enforces that jobs in the arrival sequence respect the WCET of their task, that is, for any $j \in \rho$, $\mathbf{c}(j) \leq \mathbf{C}(\tau(j))$. The execution time model is used for computing the maximum workload requested by tasks for a given duration.

Definition 6 (Task workload bound)**.** *Given a task τ and a duration Δ, the maximum workload requested by τ within that duration is*

$$wl_\tau^+(\Delta) := \mathbf{C}(\tau) \times \left\lceil \frac{\Delta}{\mathbf{P}(\tau)} \right\rceil$$

Note that $\left\lceil \frac{\Delta}{\mathbf{P}(\tau)} \right\rceil$ denotes the maximum number of task arrivals which may occur within the duration Δ.

Specification of the FPP scheduling policy

The FPP scheduling policy for uniprocessors is modeled in Prosa as two constraints on the schedule: it is *work conserving* and respects the *priority preemption*.

Definition 7 (FPP)**.** *A schedule σ over an arrival sequence ρ respects the FPP scheduling policy if and only if the two following conditions hold at any instant t:*

- (Work conserving) *If there is a pending job not scheduled at time t, then there must be another job scheduled at time t.*
- (Priority preemption) *If a job is scheduled at time t, then it has the highest priority among all pending jobs at time t.*

We use $FPP(\rho, \sigma)$ to denote that a schedule relying on an arrival sequence ρ satisfies the FPP scheduling policy.

2.4.3 Analysis

Prosa contains a RTA of a set of sporadic tasks dispatched on a uniprocessor respecting the FPP scheduling policy. The RTA provides an upper bound for arbitrary deadlines and the exact bound for implicit deadlines.

We now focus on the specification of such an analysis in Prosa. Consider a set S of sporadic tasks and a task τ_k from that set to analyze, the RTA computes the WCRT of τ_k by maximizing all workloads requested by task with a priority higher than or equal to k. The system workload bound is defined as follows:

Definition 8 (System workload bound)**.** *Given a specific task set S and a task $\tau_k \in S$ and a duration Δ, the maximum workload of the system* w.r.t. *τ_k within Δ is*

$$wl^+_{S,k}(\Delta) := \sum_{\substack{\tau_j \in S \\ j \leq k}} wl^+_{\tau_j}(\Delta)$$

The above system workload bound function has been proven correct in Prosa for any arrival sequence *w.r.t.* the set of sporadic tasks and any schedule over that arrival sequence. The workload bound $wl^+_{S,k}(\Delta)$ corresponds to the worst-case arrival pattern in which all tasks are simultaneously activated with maximum costs (corresponding to tasks' WCETs) and minimal inter-arrival separation times. It is an upper bound on the amount of service required by all tasks with a priority higher than or equal to k in any interval of size Δ. Based on this definition, a response time bound can be derived for the system model if the equation $\Delta = wl^+_{S,k}(\Delta)$ has a fixed point (Corresponding to a busy window presented in Section 2.2.2.2).

Theorem 1 (WCRT bound)**.** *Given a sporadic task set S and a task $\tau_k \in S$, then for any $R > 0$ such that $R \geq wl^+_{S,k}(R)$, any job $\jmath$ of task τ_k in an FPP schedule over an arrival sequence ρ is completed by $\mathbf{a}(\jmath) + R$.*

Note that one response time bound for a task $\tau_k \in S$ can be computed by the least positive fixed point of the function $wl^+_{S,k}(\Delta)$. It is the exact WCRT when deadlines are implicit, while it is an upper bound when deadlines are arbitrary. Using a response time bound, we can derive a *schedulability criterion* by requiring this bound to be smaller than or equal to the deadline of task τ_k.

Many real-time schedulability analyses have been formalized in Prosa. Let us cite:

- As the first case study in Prosa, Cerqueira and Brandenburg [29] formalized RTAs [19] for both the FP and EDF scheduling policies on multiprocessors as well as two extensions for released jitters and parallel jobs.
- We verified a RTA for a set of transactions with offsets executed on a uniprocessor according to the FPP scheduling policy [72]. Following this work, we [59] formalized three RTAs for FPNP and extracted a result certifier from Coq proofs. The extracted tool, itself being formally verified, is able to certify the results of CAN analyses (See Chapter 4).

- Cerqueira *et al.* [30] formalized the concepts of *strong* and *weak* suistainability. As a proof of concept, they proved that any *Job-Level Fixed Priority* scheduling policy is weakly sustainable with respect to job execution times and variable suspension times.
- Fradet *et al.* [60] provided a generic proof of *typical worst-case analysis*, an approach for analyzing weakly hard real-time systems. The authors emphasized the benefits of mechanized proofs: It provides a better understanding of the required assumptions and makes it easier to generalize existing analyses.
- Rakotomalala *et al.* [116] mechanized a subset of the Network Calculus theory. They provided some basic specifications and proofs of classic theorems in Network Calculus as well as a case study for a network of five flows using the FIFO scheduling policy.

2.4.4 Implementation

The main purpose of this part of the Prosa library is to use concrete events/programs (*e.g.,* schedulers, concrete tasks, concrete arrivals) to validate the specifications axiomatized in the system model component. For instance, to validate the specification of the FPP scheduling policy, we implement a FPP scheduler program and prove that it satisfies indeed the properties described in that specification. This component can be an interface between concrete system schedules and the proven results in Prosa in order to benefit from each other. There exist, however, some limitations to applying the proven results to a concrete system schedule due to their different basics, for instance, a concrete schedule is usually finite but it is infinite in Prosa. Chapter 3 aims at dealing with this issue.

2.5 Result Certification

There exist two main approaches to provide formal guarantees for software. An approach, called *tool verification*, is to formally verify the correctness of the full implementation of the program using a proof assistant (this can be achieved for example by using the Coq extraction mechanism to obtain a certified program). Another approach is to build a result certifier (also using a proof assistant) to validate the results produced by an uncertified commercial program. The latter approach is called *result certification*.

Tool verification has been used successfully in recent years to increase confidence in the correctness of software, especially its functional correctness. We can cite for

instance:

- The CompCert project [92] has formally verified a compiler for a significant subset of the C language using Coq. Such a certified compiler rules out the possibility of compiler-introduced bugs. The executable code produced by CompCert is proved to behave exactly as the specification of its C code.
- The seL4 microkernel [80] is the first verified OS kernel that has been formally verified for functional correctness using the Isabelle/HOL theorem prover. It can run on a variety of hardware architectures such as x86, arm and RISC-V. It has been deployed in many safety-critical projects such as the HACMS project [73] attempting at building high-assurance cyber-physical systems.
- The CertiKOS project [66] contributed a deep-specified and layer-based framework for verifying OS kernels using the Coq proof assistant. It specifies and proves correct all observed functional behaviors in order to provide as many guarantees as possible regarding the safety of an OS kernel. Using this tool, several CertiKOS kernels for a single-core processor have been verified.

Result certification has also been investigated for three main reasons:

1. Result certifications is usually very efficient;
2. For certain problems, it reduces the proof efforts compared to tool verification while providing the same guarantees;
3. It is sometimes more flexible and light-weight than tool verification. For each software update, tool verification requires adapting the proofs, while there is little or no impact when using result certification. Also, a result certifier can certify the results of uncertified commercial programs without requiring any access to their source code.

Some successful stories in result certifications are:

- As a part of the CompCert project, Rideau and Leroy [119] provided a register allocation certification algorithm using the Coq proof assistant. The authors emphasized the reduced proof effort due to using result certification. Only 900 lines of Coq code were needed for result certification, while approximately 4300 lines of Coq code were needed for tool verification, *i.e.,* to fully verify the register allocation program.
- Mabille *et al.* [100] reported an in-progress work on result certification of Network Calculus computations using Isabelle/HOL. They estimated that result certification could reduce the proof effort by a factor of 2 (or 3) compared to a fully verified Network Calculus tool.

- Armand *et al.* [10] formally verified a result checker for SAT and SMT proof witnesses using the Coq proof assistant. The checker, itself being formally verified, provides high guarantees for the witnesses produced by uncertified solvers such as ZChaff, MiniSat, or VeriT.

This chapter has allowed us to have basic and necessary information for describing the next chapters in this thesis. Chapter 3 presents how to connect the Prosa library with a verified concrete OS kernel so that they can benefit from each other. Chapter 4 shows how to certify the results of a given commercial analyzer, and Chapter 5 proposes an expressive model as well as a RTA for that model amenable to its Coq formalization in order to factorize proofs and to reduce the proof effort.

Chapter 3

Schedulability Analysis of RT-CertiKOS

Real Time Operating Systems (RTOSes) used in critical applications should have their functional and timing correctness guaranteed. Both the OS and scheduling communities have developed their own formally verified tools but there is a lack of integration between them. This chapter[1] presents an interface between a verified real-time OS kernel, namely RT-CertiKOS, built from the CertiKOS project, and a schedulability analysis from the Prosa library. Both CertiKOS and Prosa benefit from this connection: It provides a verified OS kernel with state-of-the-art schedulability analyses with reasonable proof effort; It provides Prosa with a concrete instance illustrating the practical relevance of the Prosa definitions.

Contents

[1] This is a joint work with Lionel Rieg, Maxime Lesourd, Mengqi Liu, and Zhong Shao.

3.1 Context and Motivation

Recently, the real-time and OS communities have intensified their efforts towards formal proofs, using either model checking [108, 70] or interactive theorem provers [80, 67, 29]. This trend is motivated by the high stakes involved in critical systems and the complexity of such systems, which makes pen-and-paper reasoning too error-prone.

RTOSes used in critical areas such as avionics, the automotive industry, and the medical industry must guarantee not only functional correctness but also timing requirements. For a hard real-time OS used to run critical applications, one missed deadline may have catastrophic consequences. Schedulability analysis aims to guarantee the absence of deadline misses given a scheduling policy.

Our objective is to formally verify the schedulability of a set of tasks scheduled using a concrete OS kernel. As the central part of an OS, the kernel connects software applications to system hardware. It is responsible for task management and scheduling. Therefore, in order to provide high confidence in the schedulability results, we have to verify both the functional correctness of the OS kernel and the schedulability analysis of a set of tasks executed using that kernel. In the current state of the art, the schedulability analysis is decoupled from the kernel code verification. This is good from a separation of concern perspective as both kernel verification and schedulability analysis are already complex enough without considering each other. Nevertheless, this gap also means that both communities may lack validation from each other.

On the one hand, schedulability analysis itself is error-prone. As mentioned early in Chapter 1, a flaw was found in the original schedulability analysis [147, 144, 145] for the CAN bus, which is widely used in cars. To tackle this issue and to provide high confidence in analysis results, the Prosa library [29] provides mechanized schedulability proofs. Some of its design decisions, in particular for task models and scheduling policies, are very suitable for reasoning about real-time system analyses, *e.g.,* it assumes that task arrivals and all schedule information are known and provided by two infinite traces: arrival sequences and schedules (See Chapter 2). However, it is not the case for a concrete OS kernel, *i.e.,* schedule traces of a concrete OS are finite, because we never know the future behavior. The applicability of choices made in Prosa to reality should be justified by connecting them to a concrete OS kernel enforcing a real-time scheduling policy.

On the other hand, OS kernels are very sensitive and bug-prone pieces of code. They inspire a lot of work on using formal methods to prove functional correctness and other requirements, such as access control policies [80], scheduling policies [155], timing requirements, etc. One such verified OS kernel is RT-CertiKOS [98], developed by the Yale FLINT group and built on top of sequential CertiKOS [66, 38]. Its verification focuses on extensions beyond pure functional correctness, such as real-time guarantees and isolation between components. However, any major extension such as schedulability analyses adds a lot of proof burden.

Motivation

Prosa side. The basic definitions in Prosa are good for reasoning about schedulability analyses, but their applicability to concrete OSes has not yet been tested. We want to connect Prosa to concrete OSes to make practical use of our verified analyses.

RT-CertiKOS side. Upgrading an OS kernel into a real-time one is not an easy task. When one further adds formal proofs about functional correctness, isolation, and timing requirements, the proof burden becomes enormous. Thus, from the RT-CertiKOS perspective, the benefit of using Prosa is precisely to make use of state-of-the-art schedulability analyses already certified in Coq.

Contributions

In this chapter based on [71], we solve both problems at once by combining one formal schedulability analysis given by Prosa with the functional correctness guarantees of RT-CertiKOS. Thus, we get a formal schedulability proof for this kernel: if a set of tasks satisfies the proven schedulability test from Prosa, then formal proofs ensure that there will be no deadline miss during execution on RT-CertiKOS. Furthermore, this work also produces a concrete instance of the definitions used in Prosa, ensuring their consistency and applicability to a real system.

The main contributions of this chapter are:

- The definition of a clear interface for schedulability analysis between a kernel (here, RT-CertiKOS) and a schedulability analyzer (here, the analyses of Prosa);
- A formally proven connection between RT-CertiKOS and Prosa, validating Prosa modeling choices and enabling RT-CertiKOS to benefit from the state-of-the-art schedulability results of Prosa.

Besides, this work provides solutions for addressing some technical issues when connecting RT-CertiKOS and Prosa:

- A workaround for the mismatch between the notion of jobs in schedulability analysis (which contains actual execution times) and in OS scheduling through the scheduling trace;
- A way to extend a finite scheduling trace (from RT-CertiKOS) into an infinite one (for Prosa) while still satisfying the fixed priority preemptive (FPP) scheduling policy. Note that the interface that we built is general and should work with other scheduling policies. We focus here on FPP because RT-CertiKOS uses a FPP scheduler.

3.2 The RT-CertiKOS OS Kernel

RT-CertiKOS [98], developed by the FLINT group at Yale, is a real-time extension of the single-core sequential CertiKOS [66, 38],[2] whose functional correctness has been certified in the Coq proof assistant [136]. The sequential restriction greatly simplifies the implementation of the OS kernel. However, it does not support multi-core, and the lack of kernel preemption can also degrade the responsiveness of the whole system. RT-CertiKOS proves spatial and temporal isolation (including schedulability) between components.

In this section, we first present briefly the specification and proof methodologies used for formally verifying RT-CertiKOS and then focus on the RT-CertiKOS scheduler which we want to interface with Prosa.

3.2.1 Specification and Proof Methodologies

RT-CertiKOS is organized around the notion of abstraction layers that allows decomposing the kernel into small pieces that are easier to verify.

Abstraction Layers

Abstraction layers [66] are essentially a way to combine code fragments and their interface with simulation proofs[3]. They consist of four elements: (a) a piece of code; (b) an *underlay*, the interface that the code relies on; (c) an *overlay*, the interface

[2]There is a multi-core version of CertiKOS [67, 68], but RT-CertiKOS is developed on top of the sequential version.

[3]Simulation here is a terminology in the sense of formal verification, which represents a relation between two models of which one is more precise than the other.

that the code provides; (*d*) a *simulation proof* ensuring that the code running on top of the underlay indeed provides the functionalities described in the overlay.

Both the underlay and overlay are specifications written in Coq and may be expressed using the semantics of several programming languages at once. This explains how RT-CertiKOS manages to encompass both C and assembly code verification into a unified framework. Note that this notion of interface not only includes functions but also some *abstract state*, which exposes memory states of lower layers in a clean and structured way, and allows the overlay to access them only by invoking verified functions.

Proof Methodology

RT-CertiKOS [98] follows the idea of deep specifications[4] in which the specification should be rich enough to deduce any property of interest: there should never be any need to consider the implementation. In particular, even though its source code is written in both C and assembly, the underlay always abstracts the concrete memory states it operates on into abstract states, and abstracts concrete code into Coq functions that act as executable specifications. Subsequent layers relying on this underlay will invoke Coq functions instead of the concrete code, thus hiding implementation details.

In the case of scheduling, there are essentially two functions: the scheduler and the yield system call. The scheduler relies on two concrete data structures: a counter tracking the current time and an array tracking the current budget for each task. When a job yields (*i.e.,* completed), the yield system call simply sets its remaining budget to zero. Both functions are verified in RT-CertiKOS, that is, formal proofs ensure that their C code implementations indeed simulate the corresponding Coq specifications.

3.2.2 Scheduling in RT-CertiKOS

This certified OS kernel is responsible for scheduling a set of periodic tasks.

Definition 9 (Periodic task)**.** *A periodic task τ_k is characterized by its period P_k, its WCET (or a budget) C_k, and its unique priority k.*

A task is also associated with an identifier. Tasks arrive periodically with implicit deadlines. All tasks are considered *hard*, that is, all deadlines need to be met and we assume that $0 < C_k \leq P_k$. Particularly, RT-CertiKOS enforces budgets at the

[4] https://deepspec.org/

task level: in each period, a task cannot be scheduled for more than its specified budget (*i.e.,* its WCET) and may not always use up its budget. An instance of a task may be completed without using up its budget, and the remaining budget will be freed.

The RT-CertiKOS scheduler supports the FPP scheduling policy. It maintains an integer array to keep track of budget usage for each task.

- Upon invocation, the scheduler first iterates over all tasks, replenishing budgets whenever a new period arrives. For instance, a task with its WCET 5 has a budget of 5 for each period. Within one period, its remaining budget will be relinquished after its actual job's completion (yield);
- It then loops again and finds the highest priority task, that has not used up its budget, to execute.

The scheduler is a Coq function that iterates over an abstract array of task control blocks, updates them, and returns the highest task identifier available for scheduling.

In RT-CertiKOS, a lot of the provided information that is irrelevant to schedulability analysis. Therefore, we define a simplified scheduling model of RT-CertiKOS, with a much simpler abstract state containing only the data structures that are actually used in scheduling, from which the interface data and its properties must be derived.

Definition 10 (Simplified RT-CertiKOS)**.** *The simplified abstract state contains four fields:*

ticks *the current time, that is, the number of past time slots;*

quanta *a map giving the remaining budget for each priority;*

cid *the identifier of the running task;*

schedule *the* schedule prefix*, a list remembering past* scheduling decisions *until the time* **ticks***. For any time instant, the scheduling decision is represented by either a task's identifier with its yield status (*i.e., *a boolean) or none.*

For a given schedule prefix σ_{pref}, the corresponding current time (ticks) is its length noted by $len(\sigma_{pref})$. The fact that a task τ is scheduled within σ_{pref} at time instant t is denoted as either $\sigma_{pref}(t) = (\tau, True)$ if the actual job yields (completes) at t or $\sigma_{pref}(t) = (\tau, False)$ otherwise. This scheduling model is not equivalent to the complete one, because it operates on a totally different abstract data type where all irrelevant fields are removed. Nevertheless, we still have a simulation: any step in the full RT-CertiKOS is also allowed in the simplified version and results in the

same scheduling decision and trace. To connect the full RT-CertiKOS model and the simplified one, we define a projection function *RData_proj* extracting the relevant fields from the full RT-CertiKOS state to build the simplified one. As shown in

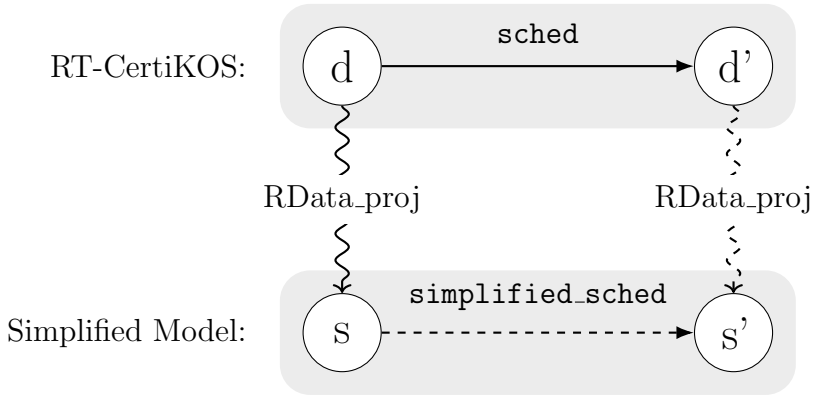


Figure 3.1: Simulation between simplified scheduling and RT-CertiKOS

Figure 3.1, we prove that given a scheduler transition of RT-CertiKOS between the (full) states d and d', there is also a transition between their projections s and s' by invoking the simplified scheduler. Formally, we prove that,

Theorem 2 (Simulation)**.** *For any state d of RT-CertiKOS,*

$$RData\_proj(sched(d)) = simplified\_sched(RData\_proj(d))$$

If the states d and s satisfy respectively the invariants for RT-CertiKOS and the simplified model, then so do d' and s' (they are invariants). As the states s and s' are projections of d and d', the invariants of s and s' also hold on the corresponding fields in d and d'. This allows us to utilize the invariants proved in the simplified model to establish properties on the full state of RT-CertiKOS.

In the following presentation, RT-CertiKOS stands for simplified RT-CertiKOS for brevity.

3.3 Connection between Prosa and RT-CertiKOS

In this section, we focus on using a proven schedulability analysis from Prosa to formally verifying the schedulability of a concrete schedule of RT-CertiKOS, which is built from a set of periodic tasks scheduled using the verified OS kernel according to the FPP scheduling policy. However, Prosa definitions cannot apply to RT-CertiKOS directly. Indeed, the perspectives of Prosa and RT-CertiKOS on the real-time aspects of a system are not the same, which is reflected in how to represent an

execution (*i.e.,* a schedule prefix in RT-CertiKOS and an infinite schedule in Prosa). We now explain how we bridge this gap to actually perform the connection.

3.3.1 Problem Statement

The key issue when connecting RT-CertiKOS with Prosa is that the proven results in Prosa rely on two infinite traces of events as shown in Figure 2.11: arrival sequences and schedules, which RT-CertiKOS does not provide. Specifically, a theorem proven in Prosa that we want to apply to RT-CertiKOS is as follows.

Theorem 3 (Schedulability analysis)**.** *Let S be a set of sporadic tasks, ρ be any infinite arrival sequence of S, and σ be any infinite schedule over ρ, then*

$$\begin{aligned}
&(\forall \tau_k \in S, R_k^+ \leq D_k) \\
&\implies \mathit{Sporadic}(\rho, S) \wedge \mathit{FPP}(\rho, \sigma) \\
&\implies (\forall \jmath \in \rho, \forall \tau_k \in S, \tau(\jmath) = \tau_k \implies R_\jmath \leq R_k^+)
\end{aligned}$$

where $R_\jmath$ is $\jmath$'s response time, R_k^+ is a τ_k's response time bound proven in Prosa for sporadic tasks scheduled with FPP, and D_k is τ_k's deadline.

Note that the basis of the theorem is the two infinite traces. In RT-CertiKOS, we can only obtain a schedule prefix due to the fact that we do not have a priori information about, *e.g.,* the execution time of jobs. Therefore, to apply Theorem 3 to the RT-CertiKOS schedule, we have to:

1. *build* an infinite arrival sequence and prove that it respects the sporadic model; and
2. *build* an infinite schedule using that arrival sequence and prove that it respects the FPP scheduling policy.

The idea of building infinite arrival sequences and schedules is that we use the schedule prefix and assume the worst-case scenarios for the future behavior. More detail is presented in the following sub-sections.

- **On the RT-CertiKOS side.** RT-CertiKOS provides a *schedule prefix* as well as its properties to the interface via its simplified model. This schedule prefix satisfies the FPP prefix property (where the FPP scheduling policy is defined based on a schedule prefix);
- **At the interface.** A job cost function is defined using a schedule prefix. The FPP scheduling policy over a schedule prefix is specified. Note that this FPP definition is different from Prosa's which is based on an infinite schedule;

- **On the Prosa side.** We build an arrival sequence using the schedule prefix and the job cost function from the interface. Based on this arrival sequence and a FPP scheduler from Prosa, we construct an *infinite schedule* with which we can apply the schedulability analysis. Then, we prove that its schedule prefix is equivalent to the one from RT-CertiKOS and is schedulable. Importantly, based on this, we prove that the subsequent executions following this schedulable prefix are schedulable.

We now present some details about the connection.

3.3.2 The RT-CertiKOS Side

Adding the schedule in RT-CertiKOS

RT-CertiKOS only maintains the current state of the system such as the current ticks and budget array. However, the interface requires a schedule trace. We introduce this variable in RT-CertiKOS, and update a few scheduling-related primitives to extend this trace whenever a task is scheduled.

This introduction adds absolutely no proof overhead, since it does not affect the scheduling decisions, thus existing proofs about the rest of the system still hold. Furthermore, it is a purely logical variable introduced through refinement, meaning that it does not exist in the C code and causes no computation overhead.

Proving the properties required by Prosa.

The interface requires two key properties:

1. the service received by each job is at most the WCET of its task; and
2. the schedule prefix satisfies the FPP scheduling policy.

These properties must be proven on the RT-CertiKOS side for any possible schedule. This way, Prosa can rely on them through the interface.

Since the RT-CertiKOS verification is based on state invariants rather than traces, we prove these properties using the following invariants:

- the length of the schedule trace is the current time + 1 (the scheduler takes a decision for the *next* time slot);
- if a task has completed in the current period, its remaining budget is 0;
- the service plus the remaining budget is equal to the job cost;
- the service received in any period is less than the WCET;

- pending jobs have positive remaining budget or equivalently, have less service than their job cost;
- the current schedule follows FPP.

To prove that these statements are indeed invariants, we must prove that they are preserved by the scheduler (triggered by the user-level timer interrupt) and by the yield system call (triggered by the user process). All other kernel steps do not modify the scheduling data of the simplified scheduling model.

3.3.3 Interface Between RT-CertiKOS and Prosa

We design an interface to link RT-CertiKOS and Prosa, focusing on the precise amount of information that needs to be transmitted between them. The interface is shaped by the information Prosa needs to perform its schedulability analysis: a task set and a schedule, together with some properties. We present the two main issues to address at the interface.

Handling service and job cost. Prosa needs, for each job, its arrival time and cost. The former can be easily defined using its task period, but the latter is unknown. In RT-CertiKOS, and more generally in any OS, we only assume a bound on the execution time of a task, used as a budget. The exact execution time of each of its jobs is not known beforehand and can be observed only at runtime. On the opposite, Prosa assumes that costs for all jobs of all tasks are part of the problem description and thus are available from the start.

To fix this mismatch, we define a job cost function computed from a schedule prefix.

Definition 11 (Job cost function)**.** *Let σ_{pref} be a schedule prefix and τ be a task, then the cost $c_{n,\sigma_{pref}}(\tau)$ of τ's n-th job is:*

$$c_{n,\sigma_{pref}}(\tau) = \begin{cases} \sum_{t=(n-1)P}^{nP} (\sigma_{pref}(t) = (\tau, \_)) & \textit{if the job completes within } \sigma_{pref}, \\ C & \textit{otherwise.} \end{cases} \tag{3.1}$$

Note that for the expression $(\sigma_{pref}(t) = (\tau, \_))$, we implicitly convert boolean values to $\{0, 1\}$ with *true* mapped to 1. Sometimes we omit σ_{pref} from $c_{n,\sigma_{pref}}(\tau)$ for brevity, *i.e.,* we denote $c_n(\tau)$ the cost of τ's n-th job. It is the service received during its n-th period by checking σ_{pref} if the job has yielded (completed) within σ_{pref}; otherwise, it is τ's WCET.

This definition relies on the computation of service in any period, which we also provide as part of the interface.

Handling infinite schedules. Prosa traces are based on an infinite schedule. RT-CertiKOS does not provide such an infinite schedule, as only a finite prefix can be known, up to the current time. Thus, we keep RT-CertiKOS's finite schedule as is in the interface and it is up to Prosa to extend it into an infinite one, suitable for its analysis.

Finally, Prosa needs two properties about the schedule prefix:

1. any task receives no more service than its WCET in any period;

2. the schedule prefix follows the FPP scheduling policy. We refer to schedules satisfying these properties as *valid schedule prefixes*.

The two properties are proved in RT-CertiKOS. The former is easy. To prove the latter, we have to specify the FPP scheduling policy over a schedule prefix (we call it FPP prefix for short). This FPP prefix is specified by an inductive definition at the interface.

Definition 12 (FPP prefix)**.** *The FPP scheduling policy is defined as an inductive predicate over a task set, an arrival sequence (not explicitly mentioned below), and a schedule prefix as follows:*

- *Any schedule on an empty trace is an FPP prefix.*
  FPP_prefix(S nil).

- *If you take any FPP prefix such that there is no job pending at its last time instant, then not scheduling any job at the next time instant will yield an FPP prefix.*
  $\forall \sigma_{pref}$, *FPP_prefix*$(S\ \sigma_{pref})$
  $\implies (\forall \tau \in S \implies \neg Pending(\tau, len(\sigma_{pref}))$
  $\implies$ *FPP_prefix*$(S\ (None :: \sigma_{pref}))$.

- *If you take any FPP prefix such that there are jobs pending at its last time instant, then one with the highest priority will be scheduled next.*
  $\forall \tau_k \in S$, $\forall \sigma_{pref}$, $\forall b$: *boolean*,
  FPP_prefix$(S\ \sigma_{pref})$
  $\implies Pending(\tau_k, len(\sigma_{pref}))$
  $\implies (\forall \tau_l \in S \implies Pending(\tau_l, len(\sigma_{pref})) \implies (k \leq l))$
  $\implies$ *FPP_prefix*$(S\ ((\tau_k, b) :: \sigma_{pref}))$.

where $Pending(\tau, t)$ *means that task* τ *is pending at time instant* t.

3.3.4 The Prosa Side

Proven schedulability analysis in Prosa.

Prosa provides a formally proven RTA as well as its corresponding schedulability criterion for a set of sporadic tasks executed on a uniprocessor according to the FPP scheduling policy. This is exactly the analysis that we want to use in RT-CertiKOS. Note that the task model used by RT-CertiKOS is periodic, which is a special case of the sporadic task model.

Consider a set S of periodic/sporadic tasks with implicit deadlines executed on a uniprocessor according to the FPP scheduling policy, and let us focus on task τ_k. Let $WCRT_k$ be the least positive fixed point of the equation $WCRT_k = wl^+_{S,k}(WCRT_k)$, where

$$wl^+_{S,k}(\Delta) := \sum_{\substack{\tau_j \in S \\ j \leq k}} wl^+_{\tau_j}(\Delta).$$

As mentioned in Theorem 3, $WCRT_k$ is a response time bound of τ_a and has been proven correct in Prosa. Note that Theorem 3 is based on the two infinite traces ρ and σ.

In order to use this response time bound, we need to relate any finite schedule prefix from the interface to an arrival sequence and a schedule satisfying the model described in Section 2.4. We can then rely on a schedulability criterion (*i.e.,* $\forall \tau_k \in S, WCRT_k \leq D_k$) to prove that the response time bound holds and deduce that any valid schedule prefix from the interface is indeed schedulable.

Bridging the gap between the interface and Prosa

The interface provides Prosa with a task set S, service and job cost functions c_n, and a valid schedule prefix σ^I_{pref} (Note that the superscript I representing interface is used to distinguish the Prosa schedule prefix, which is obtained with an infinite schedule). We first build an arrival sequence from the schedule prefix where the n-th job ($n > 0$) for a given task τ_k arrives at time $(n-1) \times P_k$ with the cost given by the interface. Note that jobs that do not arrive within the prefix cannot have yielded yet so that their costs are assumed to be the WCET of their tasks *i.e.,* we assume the worst case for the future.

Definition 13 (Concrete job)**.** *The n-th job $\jmath$ of a given task τ_k is defined by its identifier n, its task τ_k, its cost $c_n(\tau_k)$ (computed by the schedule prefix and job cost function from the interface), and its arrival time $(n-1) \times P_k$.*

Definition 14 (Concrete infinite arrival sequence)**.** *The concrete arrival sequence*

is a function ρ^c mapping any time instant t to a set of concrete jobs $\rho^c(t)$. Let $\jmath$ be a concrete job of τ_k, then

$$\jmath \in \rho^c(t) \iff (P_k \mid t) \wedge (n = \lfloor t/P_k \rfloor + 1)$$

where $P_k \mid t$ means t is multiple of τ_k's period P_k, and n is the job identifier.

For instance, at time instant t, the $(\lfloor t/P_k \rfloor + 1)$-th job of task τ_k arrives if t is multiple of P_k.

Next, we need to turn the finite schedule prefix into an infinite one. There are two possibilities: either build a full schedule from the built concrete infinite arrival sequence using the Prosa FPP scheduler, or start from the schedule prefix of the interface and extend it into an infinite one. The first technique gives for free the fact that the infinite schedule satisfies the FPP model from Prosa. The difficulty lies in proving that the schedule prefix from the interface is equivalent to the prefix of this infinite schedule. The second technique starts from the schedule prefix and the difficulty is proving that it satisfies the FPP scheduling policy as specified on the Prosa side.

The key to both two techniques is to prove the equivalence of the two FPP specifications: one, named FPP_prefix, is defined based on a schedule prefix at the interface; the other, named FPP, is specified based on an infinite schedule in Prosa (see Section 2.4). The schedule representation (*i.e.,* option job) in Prosa is different from the ones (*i.e.,* (option task, yield state)) in RT-CertiKOS.

In this thesis, we present the first strategy. A concrete schedule is built using the concrete arrival sequence and a Prosa FPP scheduler. We use a function *Prefix* that takes a Prosa schedule and a length and returns a schedule prefix with the same type as the RT-CertiKOS schedule prefix by mapping jobs to their tasks and taking care of the task yield status.

Definition 15 (Concrete infinite schedule)**.** *The concrete schedule is a function σ^c mapping any time instant t to either the concrete job scheduled at time t or $\perp$. Let $\jmath$ be a job from the concrete arrival sequence ρ^c, then*

$$\sigma^c(t) = \jmath \iff (\forall \jmath' \in \rho^c, Pending(\jmath', t) \implies p_\jmath > p_{\jmath'}) \wedge Pending(\jmath, t)$$

where $p_\jmath > p_{\jmath'}$ describes the priority of $\jmath$ is higher than that of $\jmath'$.

For each time instant t, the Prosa FPP scheduler computes all pending jobs using the concrete infinite arrival sequence and previous scheduling decisions up to

t, and selects the job with the highest priority among all pending jobs to execute at t. Obviously, the built concrete schedule σ^c satisfies *FPP*.

We prove that any prefix of the computed concrete schedule σ^c that is shorter than $len(\sigma^I_{pref})$ satisfies *FPP_prefix*.

Lemma 1 (FPP equivalence)**.** *For any length* $l \in \mathbb{N}$,

$$l \leq len(\sigma^I_{pref}) \implies FPP\_prefix(S, Prefix(\sigma^c, l)) \tag{3.2}$$

Then we prove that the schedule prefix σ^I_{pref} provided by the interface and the computed concrete infinite schedule σ^c match on the length of σ^I_{pref}. To do so, we use the fact that two FPP schedule prefixes with the same arrival sequence and scheduling policy are the same (Note that here we implicitly assume that all tasks have their own unique priority and that jobs of a task respect the FIFO order for execution).

Lemma 2 (Schedule prefix equivalence)**.**

$$Prefix(\sigma^c, len(\sigma^I_{pref})) = \sigma^I_{pref} \tag{3.3}$$

Finally, for a given task set accepted by the FPP schedulability criterion (*i.e.*, $\forall \tau_k \in S, WCRT_k \leq D_k$), we know that the computed infinite schedule is schedulable according to Theorem 3. Since its prefix is equal to the schedule prefix provided from the interface, we conclude that the schedule prefix given by the interface is schedulable.

3.4 Evaluation

As the C and assembly source code of RT-CertiKOS were not modified at all, our connection did not introduce any overhead to its performance and there is no need for a new performance evaluation. We focus on the benefits this works brings and on the amount of work involved (described in Table 3.1).

Benefits for RT-CertiKOS and Prosa

The schedulability analysis already present in RT-CertiKOS was manually proved and took around 8k LoC to handle the precise setting described in this chapter. By contrast, interfacing with Prosa requires 50% less proof code, is more flexible and can easily be extended. The introduction of a simplified scheduling model also reduced

by 75% the size of proofs of invariants about the high-level abstract scheduler since we are freed from the unnecessary information described in Section 3.3.2.

On the Prosa side, having a complete formal connection with an actual OS kernel developed independently validates the modeling choices made for describing real-time systems. Indeed, seeing schedulers as predicates over scheduling traces is very general but one can legitimately wonder whether such predicates accurately describe reality.

Proof effort

Designing a good interface allowed us to cleanly separate the work required on the RT-CertiKOS and Prosa sides.

On the RT-CertiKOS side, the design of the simplified scheduling setting was pretty straightforward, as was the correctness of the translation. Designing adequate inductive invariants to prove the two properties required by the interface was the most challenging part and unsurprisingly, it took several iterations to find correct definitions, *e.g.,* job cost function and valid schedule prefix.

On the Prosa side, building the concrete infinite arrival sequence and schedule is quite effortless given a prefix and a job cost function. The subtle thing was to find a good definition of the job cost function, which made the corresponding proofs significantly easier. Proving that the prefix of the computed infinite schedule is the same as the interface prefix was troublesome for two reasons. First, the interface prefix contains an additional boolean representing whether the scheduled job yielded and which is used for computing job costs and pending status, whereas it does not exist in the built schedule. We need this information to prove that the prefix of the computed schedule satisfies *FPP_prefix*. Second, the definition of the FPP property in the interface depends on a schedule prefix, while the one in Prosa depends on an infinite schedule.

Overall, we see the small amount of LoC required to perform this work as a validation of the applicability of our method to the considered problem.

Lessons Learned

Beyond the particular artifact linking RT-CertiKOS with Prosa, we have learned more general lessons from this connection.

First, using the same proof assistant greatly helps. Indeed, beyond the absence of technical hassle of inter-operability between different formal tools, it also avoids the pitfall of a formalization mismatch between both formal models and permits to

Table 3.1: Proof effort for connecting Prosa and RT-CertiKOS.

Feature	**Changes** (LoC)
Adding a schedule field to RT-CertiKOS	15
Interface (with proofs)	380
Simplified scheduling	100
Proving the invariants about the simplified scheduling	950
Translation RT-CertiKOS → simplified scheduling	380
Conversion between ZArith and SSReflect	280
Translation interface → Prosa	1900
Using the schedulability analysis of Prosa	130
Total	**4135**

share common definitions.

Second, the creation of an explicit interface between both tools clearly marks the flow of information, stays focused on the essential information, and delimits the "proof responsibility", that is, which side is responsible for proving which fact. It also segregates the proof techniques used on each side so as not to pollute the other one, either on a technical aspect (vanilla Coq for RT-CertiKOS *vs* the SSReflect extension for Prosa) or on the verification methods used (invariant-based properties for RT-CertiKOS *vs* trace-based properties for Prosa). This separation makes it unnecessary to have experts in both tools at once: once the interface was clearly defined, experts on each side could work with only a rough description of the other one, even though this interface required a few later changes. In particular, it is interesting to notice that this work was done partly by experts in RT-CertiKOS and the other part by experts in Prosa.

Third, the common part of the models used by both sides must be amenable to agreement. In our case, this involves having the same notion of time (scheduling slots, or ticks) and a compatible notion of schedule (finite and infinite).

Finally, we expect the interface we designed to be reusable either for other verified kernels wanting to connect to Prosa or for linking RT-CertiKOS to other formal schedulability analysis tools.

3.5 Related Work

Schedulability analysis

In order to provide formal guarantees for those results, several formal approaches have been used to verify schedulability analyses, such as model checking [54, 70, 37], temporal logic [160, 156], and theorem proving [152, 50] (See Chapter 2 for more

details).

Verification of real-time OS kernels

There is a lot of work about formal verification of OS kernels (see [81] for a survey and also Chapter 2). In this chapter, we restrict our attention to verification of real-time kernels using proof assistants. We also do not consider WCET computation, be it of the kernel itself (*e.g.,* [20, 123]) or of the task set we consider. This is a complementary but clearly distinct task to get verified time bounds.

The eChronos OS [7, 8] is a real-time OS running on single-core embedded systems. It stops its verification at the scheduling policy level, proving that the currently running task always has the highest priority among ready tasks. Xu et al. [155] verify the functional correctness of μC/OS-II [85], a real-time operating system with optimizations such as bitmaps. They also prove some high level properties, such as priority inversion freedom of shared memory IPC.

RT-CertiKOS [98] is a verified single-core real-time OS kernel developed by the FLINT group at Yale, based on sequential CertiKOS [66, 38]. It proves both temporal and spatial isolation among different components, where temporal isolation entails schedulability, etc. However, as explained in Section 3.4, its schedulability proof is long. Connecting to an existing schedulability analyzer makes it easier and more flexible.

3.6 Conclusion

Both real-time scheduling and OS communities use formal verification to provide high confidence in their theories, and they have developed their own formally verified tools but there is a lack of connection between them. This chapter presents a first step toward bridging the gap between scheduling verification and OS verification by integrating a schedulability analysis already formally proven in Prosa, with a verified sequential real-time OS kernel, RT-CertiKOS.

This has two benefits: first, it provides RT-CertiKOS with a modular, extensible, state-of-the-art formal schedulability analysis proof; second, it gives a concrete instance of one of the scheduling theories described in Prosa, thus ensuring that its model is consistent and applicable to actual systems. It is a validation of Prosa decisions and shows that schedulability analyses proven in the convenient setting of infinite traces can still be applied to lower level models of real-time systems with finite traces. We believe this connection can be easily adapted to other verified kernels or schedulability analyzers.

It also showcases that it is possible and practical to connect two completely independent medium- to large-scale formal proof developments between two communities.

Chapter 4

Certification for CAN Analyses

In order to provide high confidence in CAN real-time properties, this chapter presents several formally proven RTAs for CAN based on Tindell's transaction with offsets task model. Particularly, the chapter provides an optimized RTA combining the two most widely used CAN analyses, which is then used in CertiCAN, a tool formally specified and proved in Coq for certifying CAN analysis results.

CertiCAN first tries to certify the result provided as input using the approximate analysis only, then resorts to a more precise analysis for cases which cannot be certified using the approximate analysis. Experiments demonstrate the potential of CertiCAN as well as the corresponding certified analyzer for industry practice. It is able to certify the results returned by RTaW-Pegase even for large systems.

Contents

4.1 Context and Motivation

A recent series of mistakes in the analysis of self-suspending tasks [21] underlines the limitations of pen-and-paper proofs for such complex problems. This issue is not new, as illustrated by the flaw in the original Response Time Analysis (RTA) of CAN messages proposed by Tindell *et al.* [144, 147, 145], which was found and fixed many years later [43]. This motivated us to build certified[1] proofs for real-time systems analysis in academic research. A second motivation behind the need to certify real-time systems analysis results comes from industry. Standards such as ISO 26262 for automotive or DO-178C for avionics advocate the use of formal methods for the development and validation of safety critical systems. Therefore, we aim at providing certified real-time guarantees for industrial systems.

In this chapter, we focus on the CAN analysis cited earlier [144]. The underlying analysis is a RTA for task sets with offsets under the FPNP scheduling policy, with a notion of transaction, *i.e.,* messages sent from the same Electronic Control Unit (ECU). The CAN [127] protocol is widely used in automotive applications and there exist several commercial tools performing CAN analysis. Among these, we focus on RTaW-Pegase [110], for which we obtained an academic license.

[1]Throughout the thesis, the term *certified* means *formally verified* using a proof assistant, in our case Coq.

Certifying the results of a CAN analysis tool

Rather than certifying RTaW-Pegase, that is, formally proving that the CAN analysis implemented in RTaW-Pegase is correct, we choose to build a tool based on the Coq proof assistant that can certify the results of the CAN analysis performed by RTaW-Pegase. In other words, our tool, called CertiCAN, can be called every time a result obtained with RTaW-Pegase[2] must be certified. This choice is motivated by the fact that result certification is a process that is light-weight and flexible compared to tool certification, which makes it a practical choice for industrial purposes. Indeed, RTaW-Pegase is a complex tool for which we do not have the source code. It is likely to be highly optimized and subject to regular changes. All this would make it difficult to certify the tool directly and this correctness proof would need to be updated regularly.

Our problem is then: Can we certify efficiently enough the analysis results computed by RTaW-Pegase? Compared to a traditional RTA, can we use the fact that a result certifier is given as input the bound it is expected to certify?

Our solution is based on the following idea: We use a combination of two existing analysis techniques, one precise but with high computational complexity and another one much faster but approximate (it may compute pessimistic upper bounds on response times). These two analyses were introduced by Tindell for the RTA of tasks with offsets scheduled according to the Fixed Priority Preemptive policy [142, 143]. The precise analysis was adapted to CAN by Meumeu Yomsi *et al.* [159].

CertiCAN combines in an optimized way the two analyses. It first tries to certify the result provided as input using the approximate analysis only, then resorts to a more precise analysis for cases which cannot be certified using the approximate analysis. Experiments demonstrate the potential of CertiCAN for industry practice. It is able to certify the results returned by RTaW-Pegase even for large systems.

Contribution

The main contribution of this chapter is CertiCAN, the first formally proven tool able to certify the results of commercial CAN analysis tools. This is however not the only contribution of the chapter. More specifically, we propose:

1. A new RTA for CAN that combines two well-known analyses, one precise and another approximate;
2. The correctness proof in Coq of the three analyses;

[2]Note that CertiCAN does not depend on the internals of the RTA tool considered. It is interoperable with any other tool analyzing the same CAN system models as RTaW-Pegase.

3. Three Coq-certified tools in OCaml extracted from the proofs, one for each analysis;
4. Based on the same principle as the new RTA, a method and its corresponding tool formally verified in Coq; The tool, called CertiCAN, is intended to certify the results of non certified tools such as RTaW-Pegase.
5. Proved optimizations for CertiCAN's efficiency;
6. Experiments that demonstrate the usability of CertiCAN as well as its corresponding certified analyzer for industry practice.

Beyond CertiCAN, we believe that the results presented in this chapter are significant in that they demonstrate the advantage of result certification over tool certification for the RTA of CAN buses. In addition, the underlying technique can be reused for any other system model for which there exist RTAs with different levels of precision.

All the Coq specifications and proofs are available online [32].

4.2 Controller Area Network

The CAN network is a vehicle communication bus which is widely used in many industrial domains, in particular, in the automotive industry. In critical applications, it is essential to perform RTAs in order to ensure that systems can meet their timing requirements *i.e.,* the response time of a message (*i.e.,* a task) is smaller than its deadline.

The analyses proposed by RTaW-Pegase are based on a precise RTA of periodic tasks with offsets dispatched according to the FPNP scheduling policy [159]. In addition to the precise analysis, RTaW-Pegase proposes an approximate but faster version. The implementation of these analyses uses several undocumented optimizations.

In this section, first of all, we briefly present some notions of the CAN protocol which are useful for describing a CAN system. We refer interested readers to the official CAN specification [127] for more detail; Then, we present the system model considered in these analyses as well as notations and definitions used throughout this chapter.

4.2.1 The CAN protocol

The CAN protocol describes communication rules between ECUs on a CAN bus. The topology of a CAN bus is illustrated in Figure. 4.1. Each ECU uses a local

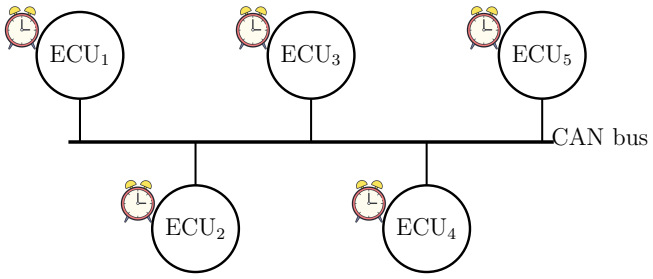


Figure 4.1: An example of CAN bus

clock and sends messages to other ECUs according to the FPNP scheduling policy. Each message is assigned a unique priority and cannot be preempted once it has started transmission. When several messages are waiting for transmission, the bus chooses the message with the highest priority. Messages to be sent are encapsulated in a fixed frame. The frame consists of:

- a unique identifier (11 bits for the standard format and 29 bits for the extended format);
- a message to transfer, whose size is noted m. It could be 1 to 8 bytes; and
- other control bits *e.g.,* acknowledgment, error detection, bit stuffing, *etc.*

In the worst case, the size of a frame is $55 + 10m$ bits for the standard format and $80 + 10m$ bits for the extended format [43].

Maximum transmission time

Usually, the bit rate of a CAN bus is fixed, which can be one of the following: 125 kbit/s, 250 kbit/s, 500 kbit/s, 1 Mbit/s. With the above information, we can determine the maximum transmission time of a message. For instance, on a 500 kbit/s CAN bus, the transfer of an 8 byte message using the standard frame takes

$$\frac{(55 + 10 * 8)\text{ bits}}{500\text{ kbit/s}} = 270\ \mu\text{s}$$

This formula is used later to produce task sets for experimental evaluations.

Response time

A message may not be immediately transmitted after its activation (*i.e.,* its request to send). It is delayed until the bus has sent all higher priority pending messages. The response time of a message is defined as the time duration between its activation and its completion (that is, the end of its transmission via the CAN bus). It is usually

associated with a real-time constraint stating that its response time should be less than a given duration *i.e.,* its deadline.

Fixed timing relation between messages within one ECU

Each ECU communicates with other ECUs through several periodic messages. All activation times of a message are known once its first activation time is fixed. To reduce interference between different messages, messages activations are separated by introducing a so-called *offset*, parameter for each message. An offset is a fixed time duration from local time 0 to the first activation time of a message. Introducing offsets helps to increase the bus utilization while keeping the system schedulable.

4.2.2 System model

The system model considered consists of a set of transactions representing ECU nodes

$$Sys := \{ Tr_1, Tr_2, \ldots, Tr_N \}$$

where each transaction Tr_i is a set of periodic tasks (representing messages):

$$Tr_i := \{\tau_{i,1}, \tau_{i,2}, \ldots, \tau_{i,M}\}$$

Each task $\tau_{i,k}$ has a fixed and unique priority k (a smaller number means a higher priority) and is characterized by a 4-tuple

$$(C(\tau_{i,k}), D(\tau_{i,k}), P(\tau_{i,k}), O(\tau_{i,k}))$$

where

- $C(\tau_{i,k})$ denotes its worst-case execution time (WCET), *i.e.,* the maximum transmission time,
- $D(\tau_{i,k})$ its relative deadline,
- $P(\tau_{i,k})$ its activation period, and
- $O(\tau_{i,k})$ its *offset*, *i.e.,* the delay between the first release of its transaction Tr_i and the first activation of the task. In this chapter, *constrained* offsets are assumed, formally $O(\tau_{i,k}) < P(\tau_{i,k})$.

Tasks within the same transaction share the same clock. All tasks of Tr_i being periodic, their offsets define a precise timing relation between them.

Task $\tau_{i,k}$ activates periodically its jobs at $O(\tau_{i,k}) + m.P(\tau_{i,k})$ with $m \geq 0$. A job $\jmath$ of a task $\tau_{i,k}$ is characterized by

- its activation time $a_{i,k}(\jmath)$,
- its completion time $end_{i,k}(\jmath)$ and
- its cost $c_{i,k}(\jmath)$ $(c_{i,k}(\jmath) \leq C(\tau_{i,k}))$

Its response time $R_{i,k}(\jmath)$ is defined as $end_{i,k}(\jmath) - a_{i,k}(\jmath)$. The worst-case response time (WCRT) of task $\tau_{i,k}$, denoted $wcrt_{i,k}$, is the largest possible response time among all jobs of task $\tau_{i,k}$.

The model does not suppose any global synchronization between transactions. Any possible time shift between any two transactions is assumed to be possible and must be considered by the analysis.

4.2.3 Notations and definitions

As in previous chapters, we note $hep(k)$, $hp(k)$, $lp(k)$ the sets of tasks of the system under study whose priorities are higher than or equal to, higher than or lower than k, respectively.

The RTAs considered here rely on the concept of busy window presented in Section 2.2.2.2, which we formally define now.

Definition 16 (Level-k quiet time)**.** *An instant t is said to be a* level-k quiet time *if all jobs of priority higher than or equal to k released strictly before t have completed at t.*

Definition 17 (Level-k busy window)**.** *A time interval $[t_1, t_2[$ is said to be a* level-k busy window *if:*

1. *t_1 and t_2 are level-k quiet times;*
2. *there is no level-k quiet time in $]t_1, t_2[$; and*
3. *at least one job with a priority higher than or equal to k is released in $[t_1, t_2[$.*

Clearly, a job with a priority higher than or equal to k has completed by the end of its level-k busy window. In other words, its response time can be bounded by the length of the corresponding busy window. Such a bound is however quite coarse. In particular, there may be several jobs of the same task activated in the same busy window. We thus consider the response time of each level-k job in a level-k busy window. To this aim, we use the notions of phase and queueing prefix as defined in [58] and informally defined in Section 2.2.2.2.

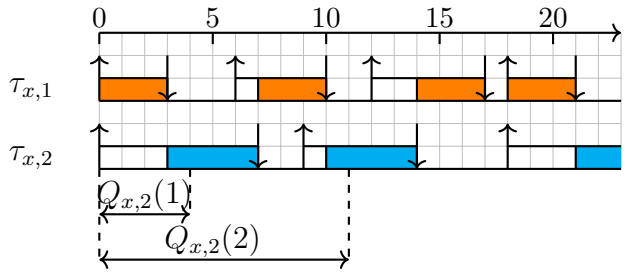


Figure 4.2: Example of queueing prefixes in a busy window.

Definition 18 (Queueing prefix)**.** *The* q-th queueing prefix *of task* $\tau_{i,k}$ *in a level-*k *busy window* $[t_1, t_2[$ *is the time interval* $[t_1, t_q]$ *where* t_q *is the instant at which the* q*-th job of task* $\tau_{i,k}$ *receives its first service (*i.e., *is scheduled for the first time).*

Figure 4.2 shows the first and second queueing prefixes of a task $\tau_{x,2}$ in a level-2 busy window, $Q_{x,2}(1)$ and $Q_{x,2}(2)$ respectively.

Definition 19 (Phase)**.** *The* phase *of the* q*-th job* $\jmath$ *of task* $\tau_{i,k}$ *in a level-*k *busy window* $[t_1, t_2[$ *is the duration* $a_{i,k}(\jmath) - t_1$*.*

As mentioned in Chapter 2, due to the FPNP scheduling policy, a lower priority job than k can be executed at the beginning of a level-k busy window. This is referred to as the *blocking factor*. It is easy to prove that the blocking factor of a level-k busy window is bounded by:

$$B_k = \max_{\tau_{i,x} \in lp(k)} (C(\tau_{i,x}) - \epsilon) \tag{4.1}$$

where $\epsilon = 1$ (*i.e.,* a time unit). Indeed, the worst case is when the lower priority task with the largest worst-case execution time activates a job with such an execution time just one time unit before the start of the level-k busy window.

Another key notion for RTA is the workload of a task, which quantifies its request for resources.

Definition 20. *The maximum workload of task* $\tau_{i,k}$ *for a given interval* $[t_1, t_1 + \Delta[$ *is defined as:*

$$wl^+_{\tau_{i,k}}(t_1, \Delta) = (\underbrace{\lceil \frac{\Delta - \theta_{i,k}(t_1)}{P(\tau_{i,k})} \rceil}_{n_a}) C(\tau_{i,k}) \tag{4.2}$$

where $\theta_{i,k}(t_1) = (P(\tau_{i,k}) + O(\tau_{i,k}) - (t_1 \bmod P(\tau_{i,k}))) \bmod P(\tau_{i,k})$ *is the time duration between* t_1 *and the first activation of* $\tau_{i,k}$ *after* t_1*, and* n_a *is the maximum number of activations of* $\tau_{i,k}$ *during the duration* $\Delta - \theta_{i,k}(t_1)$*.*

(a) $(O(\tau_{i,k}) \geq \phi_i)$. **(b)** $(O(\tau_{i,k}) < \phi_i)$.

Figure 4.3: Illustrations of $\theta_{i,k}(t_1)$ in two cases ($\star = a_{i,k}(\jmath)$, $\phi_i = t_1 \ mod \ P(\tau_{i,k})$, and $\uparrow$ represents releases of transaction Tr_i).

When the tasks of the same transaction have distinct periods, the RTA presented in the next section makes use of the so-called hyper-period of transactions.

Definition 21. *(Hyper-period). The hyper-period T_i^+ of a transaction $Tr_i := \{\tau_{i,1}, \tau_{i,2}, \ldots, \tau_{i,M}\}$ is the least common multiple of the periods of all its tasks. Formally,*

$$T_i^+ = lcm\{P(\tau_{i,1}), P(\tau_{i,2}), \ldots, P(\tau_{i,M})\} \tag{4.3}$$

4.3 RTAs for CAN

In this section, we describe the RTAs for CAN that we use to certify the results of the RTaW-Pegase tool. The correctness of these RTAs has been proved using the Coq proof assistant [138] on top of the Prosa library [112]. In the following, we consider a task $\tau_{i,k}$ and describe how the two RTAs compute an upper bound on its worst-case response time. The presentation follows the Coq specification. We omit proofs of lemmas and theorems and refer the interested reader to the Coq source [32].

The two RTAs follow the same procedure:

1. Let $\jmath$ be a job of task $\tau_{i,k}$.
2. For all possible *scenarios*[3] (precise or approximate) compute an upper bound $R_{BW_k}(\jmath)$ of that job $\jmath$ by examining all level-k queueing prefixes in its corresponding worst-case level-k busy window $BW_k(\jmath)$.
3. The upper bound on the response time of $\jmath$, denoted by $R_{BW_k}(\jmath)$, is the maximum of the results for all scenarios.

Both analyses rely heavily on the analysis of busy windows and queueing prefixes introduced in the previous section.

[3] A scenario represents a specific alignment among transactions that corresponds to a set of maximum workload functions, which is used to compute an upper bound on the response time of $\tau_{i,k}$ for that alignment. A scenario is said *approximate* if some maximum workload functions are approximated.

4.3.1 Busy window analysis

We start by describing how to analyze the response time of a task $\tau_{i,k}$ in a concrete busy window. Assume that there exists a level-k busy window $[t_1, t_2[$ in which a job $\jmath$ of $\tau_{i,k}$ is released. It can be shown that if the utilization is below 100%, it is always possible to compute that busy window. Assume that this job is the q-th job of task $\tau_{i,k}$ arrived in the busy window $[t_1, t_2[$. Let $Q_{i,k}(q)$ denote the q-th queueing prefix of task $\tau_{i,k}$ (see Def. 18) in that busy window, then $\jmath$ finishes at the latest at

$$t_1 + Q_{i,k}(q) + c_{i,k}(\jmath) - 1$$

The response time of this instance is bounded by:

$$R_{i,k}(\jmath) \leq t_1 + Q_{i,k}(q) + c_{i,k}(\jmath) - 1 - a_{i,k}(\jmath)$$

The phase of $\jmath$ (see Def. 19) can also been defined as

$$a_{i,k}(\jmath) - t_1 = \theta_{i,k}(t_1) + (q-1) * P(\tau_{i,k})$$

As a result, the bound of the response time of job $\jmath$ can be rewritten as

$$R_{i,k}(\jmath) \leq Q_{i,k}(q) - \underbrace{(\theta_{i,k}(t_1) + (q-1) * P(\tau_{i,k}))}_{\text{phase}} + c_{i,k}(\jmath) - 1$$

Let us write $BW_k = t_2 - t_1$ to denote the size of the busy window[4]. We know that there are at most

$$q^+_{\tau_{i,k}, BW_k} = \left\lceil \frac{BW_k}{P(\tau_{i,k})} \right\rceil$$

jobs of task $\tau_{i,k}$ in that busy window. Therefore, within busy window $[t_1, t_2[$, the WCRT of task $\tau_{i,k}$ can be locally bounded by $R_{BW_k}(\jmath)$ defined as

$$\max_{q \leq q^+_{\tau_{i,k}, BW_k}} (Q_{i,k}(q) - \underbrace{(\theta_{i,k}(t_1) + (q-1) * P(\tau_{i,k}))}_{\text{phase}} + c_{i,k}(\jmath_q) - 1)$$

where $\jmath_q$ represents the q-th job of task $\tau_{i,k}$ released in the busy window.

To find the WCRT of task $\tau_{i,k}$, we must find, for any possible scenario,

1. an upper bound on BW_k;
2. an upper bound on $Q_{i,k}(q)$ for any $q \leq q^+_{\tau_{i,k}, BW_k}$;

[4]Note that we denote BW_k the size of the busy window that starts with t_1 and omit t_1 from BW_k for brevity. Similarly for the following notation $Q_{i,k}(q)$.

3. a lower bound on $\theta_{i,k}(t_1)$.

BW_k and $Q_{i,k}(q)$ are defined as fixed points of two workload functions. In order to define these functions, we must first define the notion of workload. Note that this definition is different from the maximum workload function defined in Section 4.2.

Definition 22 (workload)**.** *The* workload $wl\tau_{j,l}(t_1, \Delta)$ *of task* $\tau_{j,l}$ *in a time interval* $[t_1, t_1 + \Delta[$ *is the cumulative cost (*i.e., *required service time) of its jobs released in that interval.*

BW_k can be found by computing the least fixed point of the following equation:

$$BW_k = f_B(BW_k)$$

where

$$f_B(\Delta) = b_k(t_1, \Delta) + \sum_{\substack{\tau_{j,l} \in Tr_j \\ Tr_j \in Sys \\ l \leq k}} wl_{\tau_{j,l}}(t_1, \Delta)$$

and $b_k(t_1, BW_k)$ is the blocking factor, that is the time duration at the beginning of the busy window when a lower priority task may execute.

Similarly, $Q_{i,k}(q)$ can be found by computing the least fixed point of the following equation:

$$Q_{i,k}(q) = f_Q(q, Q_{i,k}(q))$$

where

$$\begin{aligned} f_Q(q, Q_{i,k}(q)) \quad &= b_k(t_1, \Delta) + \sum_{\substack{\tau_{j,l} \in Tr_j \\ Tr_j \in Sys \\ l < k}} wl_{\tau_{j,l}}(t_1, \Delta) \\ &+ \; wl_{\tau_{i,k}}\left(t_1, \theta_{i,k}(t_1) + (q-1) * P(\tau_{i,k})\right) + 1 \end{aligned}$$

A static analysis will have to find upper bounds for any possible level-k busy window. To this aim, it is sufficient to find two functions that bound $f_B(\Delta)$ and $f_Q(q, \Delta)$ and to compute their fixed points. The correctness of this approach is expressed by the following lemma.

Lemma 3. *Let* $f, g : \mathbb{N} \to \mathbb{N}$ *be two monotonically increasing functions and* Δ_1 *and* Δ_2 *be fixed points of the equations* $\Delta = f(\Delta)$ *and* $\Delta = g(\Delta)$ *then, if for all* $x : \mathbb{N}$, $f(x) \leq g(x)$ *and, for all* $x : \mathbb{N}^+$, $x < \Delta_1$, *we have* $x < f(x)$, *then* $\Delta_1 \leq \Delta_2$.

Finally, computing an upper bound on the WCRT of task $\tau_{i,k}$ amounts to finding a finite set of scenarios such that $f_B(\Delta)$ and $f_Q(q, \Delta)$ for any busy window are bounded by the corresponding functions of a scenario in that set. The WCRT of the task $\tau_{i,k}$ is found by taking the maximum WCRT found for all these scenarios.

4.3.2 Precise analysis

The precise analysis considers the finite set of scenarios corresponding to the cases where

1. all jobs in the busy window take their worst-case execution time to complete; and
2. t_1 is aligned with an activation in each transaction.

The set of possible alignments corresponds to the set of scenarios.

We must show that, for any concrete busy window starting at t_1, there is a scenario belonging to the set described above that maximizes the functions $f_B(\Delta)$ and $f_Q(q, \Delta)$ and minimizes $\theta_{i,k}(t_1)$.

First, we show a lower bound of $\theta_{i,k}(t_1)$ (*i.e.,* the time duration between t_1 and the first activation of $\tau_{i,k}$ after t_1).

We prove that $\theta_{i,k}$ decreases if we right shift t_1 to t'_1 where the first activation with a priority higher than or equal to k in Tr_i occurs after t_1, *i.e.,* $\theta_{i,k}(t'_1) \leq \theta_{i,k}(t_1)$.

Let us denote $O_i^{t_1}$ the duration between the latest release of transaction Tr_i before t_1 and the first activation with a priority higher than or equal to k in Tr_i after t_1. As presented in Definition 20, $\theta_{i,k}$ is computed by using the modulo operation. Due to the modulo identity property, we have $\theta_{i,k}(t'_1) = \theta_{i,k}(O_i^{t_1})$, therefore,

Lemma 4 (Alignment-θ)**.**

$$\theta_{i,k}(O_i^{t_1}) \leq \theta_{i,k}(t_1) \tag{4.4}$$

Next, let us compute upper bounds of functions $f_B(\Delta)$ and $f_Q(q, \Delta)$. To this aim we bound the blocking time and the workload.

It is easy to prove that the actual blocking time in a level-k busy window is bounded by B_k (see Equation. 4.1):

Lemma 5. $b_k(t_1, BW_k) \leq B_k$

For any task $\tau_{j,l}$ in the system, and for any time instant t_1 and time duration Δ, the workload of task $\tau_{j,l}$ is maximized when all its jobs take their WCET (see Def. 20). Formally,

Lemma 6 (WCET)**.** $wl_{\tau_{j,l}}(t_1, \Delta) \leq wl^+_{\tau_{j,l}}(t_1, \Delta)$

For any task $\tau_{j,l}$ in the system, and for any time instant t_1 and time duration Δ, the workload of task $\tau_{j,l}$ is maximized when we right shift the interval $[t_1, t_1 + \Delta[$ to align t_1 with the first activation with a priority higher than or equal to k in Tr_j after t_1.

Lemma 7 (Alignment-wl)**.** $wl^+_{\tau_{j,l}}(t_1, \Delta) \leq wl^+_{\tau_{j,l}}(O_j^{t_1}, \Delta)$

With the three above lemmas, we can provide upper bounds to the functions $f_B(\Delta)$ and $f_Q(q, \Delta)$.

Lemma 8 (Bound-$f_B(\Delta)$)**.** *For any time duration* Δ*,*

$$f_B(\Delta) \leq f_B^+(\Delta) \tag{4.5}$$

where

$$f_B^+(\Delta) := B_k + \sum_{Tr_j \in Sys} wl^+_{Tr_j}(O_j^{t_1}, \Delta) \tag{4.6}$$

and

$$wl^+_{Tr_j}(O_j^{t_1}, \Delta) = \sum_{\substack{\tau_{j,l} \in Tr_j \\ l \leq k}} wl^+_{\tau_{j,l}}(O_j^{t_1}, \Delta) \tag{4.7}$$

Lemma 9 (Bound-$f_Q(q, \Delta)$)**.** *In any level-k busy window* BW_k *and for any time duration* Δ

$$f_Q(q, \Delta) \leq f_Q^+(q, \Delta) \tag{4.8}$$

where

$$\begin{aligned} f_Q^+(q, \Delta) := B_k & \\ & + \sum_{\substack{Tr_j \in Sys \\ j \neq i}} wl^+_{Tr_j}(O_j^{t_1}, \Delta) \\ & + \sum_{\substack{\tau_{i,h} \in Tr_i \\ h < k}} wl^+_{\tau_{i,h}}(O_i^{t_1}, \Delta) \\ & + wl^+_{\tau_{i,k}}\left(O_i^{t_1}, \theta_{i,k}(O_i^{t_1}) + (q-1) * P(\tau_{i,k})\right) + 1 \end{aligned} \tag{4.9}$$

Let LO^{t_1} be a list of alignments which consists of one $O_j^{t_1}$ for each transaction $Tr_j \in Sys$ and let BW_k^+ be the least fixed point of the following equation:

$$BW^+_{LO^{t_1}} = f_B^+(BW^+_{LO^{t_1}}) \tag{4.10}$$

For any $q \leq q^+_{BW^+_{LO^{t_1}}}$, we compute the least fixed point of equation:

$$Q^+_{LO^{t_1}} = f_Q^+(q, Q^+_{LO^{t_1}}) \tag{4.11}$$

Then, the response times of jobs of task $\tau_{i,k}$ released in the busy window $[t_1, t_2[$

are upper bounded by $RT^{+}_{LO^{t_1}}(\tau_{i,j})$ defined as:

$$\max_{q \leq q^{+}_{BW^{+}_{LO^{t_1}}}} \left\{ Q^{+}_{LO^{t_1}}(q) - \underbrace{(\theta_{i,k}(O_i^{t_1}) + (q-1) * P(\tau_{i,k}))}_{\text{phase}} + C(\tau_{i,k}) - 1 \right\} \tag{4.12}$$

To upper bound the WCRT $wcrt_{i,k}$ of the task $\tau_{i,k}$, we need to test all possible such $O_j^{t_1}$ for each transaction Tr_j. The list LO_j of candidates for $O_j^{t_1}$ for each transaction Tr_j is composed of all possible activations of jobs with a higher priority than k ($\in hep(k)$) in the transaction Tr_j within its hyper-period T_j^{+}:

$$LO_j = \bigcup_{\tau_{j,l} \in Tr_j \cap hep(k)} \{o \mid o = O(\tau_{j,l}) + x * P(\tau_{j,l}), o < T_j^{+}, x \in \mathbb{N}\}$$

This list represents all possible alignments of the busy window with an activation. The list of all scenarios is made of all combinations of alignments over all transactions. The WCRT $wcrt_{i,k}$ of task $\tau_{i,k}$ is bounded by the maximal WCRT of all scenarios. Formally,

Theorem 4. *Let $\times$ denote the cartesian product, then*

$$wcrt_{i,k} \leq \max_{o \in LO_1 \times \cdots \times LO_N} RT^{+}_{o}(\tau_{i,k})$$

4.3.3 Approximate analysis

For large systems, the number of precise scenarios explodes and the precise analysis quickly becomes intractable. In this subsection, we present a more efficient but approximate analysis. It follows the same approach as presented in [142]. Its principle is to use the approximate scenarios, which consist in the alignments of the considered transaction Tr_i only; the other transactions are represented by the approximate workload bound function.

The approximate workload bound function of a transaction is to maximize the workload among all possible alignments and is defined as follows.

Definition 23. *The approximate workload bound function of a transaction Tr_j for the duration Δ is defined as the maximum workload among all possible alignments represented by LO_j:*

$$wl^{*}_{Tr_j}(\Delta) = \max_{o \in LO_j} \left\{ \sum_{\substack{\tau_{j,l} \in Tr_j \\ l \leq k}} wl^{+}_{\tau_{j,l}}(o, \Delta) \right\}$$

Functions $f_B^+(\Delta)$ and $f_Q^+(q,\Delta)$ are upper bounded by using $wl^*_{Tr_j}(\Delta)$ for each transaction Tr_j. However, in order to obtain a tighter bound, we compute the precise workload of transaction Tr_i of task $\tau_{i,k}$ (*i.e.,* the task we analyze).

Lemma 10 (Bound-$f_B^+(\Delta)$)**.** *For any time duration* Δ *and any* $o \in LO_i$

$$f_B^+(\Delta) \leq f_B^*(\Delta)$$

where

$$f_B^*(\Delta) := B_k + \sum_{\substack{Tr_j \in Sys \\ j \neq i}} wl^*_{Tr_j}(\Delta) + \sum_{\substack{\tau_{i,l} \in Tr_i \\ l \leq k}} wl^+_{\tau_{i,l}}(o,\Delta)$$

Lemma 11 (Bound-$f_Q^+(q,\Delta)$)**.** *For any time duration* Δ

$$f_Q^+(q,\Delta) \leq f_Q^*(q,\Delta)$$

where

$$\begin{aligned} f_Q^*(q,\Delta) := B_k & \\ & + \sum_{\substack{Tr_j \in Sys \\ j \neq i}} wl^*_{Tr_j}(\Delta) \\ & + \sum_{\substack{\tau_{i,h} \in Tr_i \\ h < k}} wl^+_{\tau_{i,h}}(O_i^{t_1},\Delta) \\ & + wl^+_{\tau_{i,k}}\left(O_i^{t_1}, \theta_{i,k}(O_i^{t_1}) + (q-1) * P(\tau_{i,k})\right) + 1 \end{aligned} \tag{4.13}$$

We compute $BW^*_{O_i^{t_1}}$ the least fixed point of equation

$$BW^*_{O_i^{t_1}} = f_B^*(BW^*_{O_i^{t_1}})$$

and, for each $q \leq q^+_{BW^*_{O_i^{t_1}}}$, the least fixed point of equation:

$$Q^*_{O_i^{t_1}} = f_Q^*(q, Q^*_{O_i^{t_1}})$$

then, the response time of jobs of task $\tau_{i,k}$ released in the busy window $[t_1, t_2[$ is upper bounded by $RT^*_{O_i^{t_1}}(\tau_{i,k})$ defined as:

$$\max_{q \leq q^+_{BW^*_{O_i^{t_1}}}} \left\{ Q^*_{O_i^{t_1}}(q) - \underbrace{\left(\theta_{i,k}(O_i^{t_1}) + (q-1) * P(\tau_{i,k})\right)}_{\text{phase}} + C(\tau_{i,k}) - 1 \right\}$$

Then, the WCRT $wcrt_{i,k}$ of task $\tau_{i,k}$ is the maximum of these values for all possible alignments represented by LO_i.

Theorem 5.

$$wcrt_{i,k} \leq \max_{o \in LO_i} RT^*_o(\tau_{i,k})$$

Compared to the precise analysis, we do not consider all possible combinations (the cartesian product) of all alignments of all transactions.

4.3.4 Generic Analysis

We have presented two analyses for CAN:

- A precise analysis which provides a precise result but quickly becomes intractable.
- An approximate analysis which is able to efficiently return approximate results.

To benefit from the advantages of both the precise analysis and the approximate analysis, we combine the two analyses together in order to use as much as possible the approximate version to compute the precise results. The combined analysis presented in Section 4.4.3 relies on a generic analysis that we want to present now. The generic analysis is a generic version of the two analyses: precise and approximate. It separately computes precise workloads and approximate workloads. Transactions in systems are divided into two disjoint sets:

1. SET_p (Contributing precise workload); Each transaction in SET_p provides a precise workload for each specific alignment between transactions among this set. All possible alignments are considered to perform the worst-case response time for a given task $\tau_{i,k}$ to be analyzed. Note that the transaction Tr_i containing the task under consideration $\tau_{i,k}$ is considered to be in this set.
2. SET_a (Contributing approximate workload). Each transaction in SET_a provides an approximate workload as defined in Definition 23.

Considering the two sets, functions $f^+_B(\Delta)$ and $f^+_Q(q, \Delta)$ can be upper bounded by using $wl^+(\Delta)$ for transactions from SET_p and $wl^*(\Delta)$ for transactions from SET_a.

Lemma 12 (Refined Bound-$f^+_B(\Delta)$)**.** *For any time duration Δ and any $\tilde{O} \in \tilde{LO}_{\text{SET}_p} := \{\cdots \times LO_p \times \cdots\}$ where LO_p is the set of candidates for transaction $Tr_p \in \text{SET}_p$,*

$$f^+_B(\Delta) \leq \tilde{f}^+_B(\Delta)$$

where

$$\tilde{f}_B^+(\Delta) := B_k + \sum_{Tr_a \in \text{SET}_a} wl^*_{Tr_a}(\Delta) + \sum_{\substack{Tr_p \in \text{SET}_p \\ o_p \in \tilde{O}}} wl^+_{Tr_p}(o_p, \Delta)$$

Lemma 13 (Refined Bound-$f_Q^+(q,\Delta)$)**.** *For any time duration* Δ

$$f_Q^+(q,\Delta) \leq \tilde{f}_Q^+(q,\Delta)$$

where

$$\begin{aligned}\tilde{f}_Q^+(q,\Delta) := B_k & \\ &+ \sum_{Tr_a \in \text{SET}_a} wl^*_{Tr_a}(\Delta) \\ &+ \sum_{\substack{Tr_p \in \text{SET}_p \\ p \neq i}} wl^+_{Tr_p}(O_p^{t_1}, \Delta) \\ &+ \sum_{\substack{\tau_{i,h} \in Tr_i \\ h<k}} wl^+_{\tau_{i,h}}(O_i^{t_1}, \Delta) \\ &+ wl^+_{\tau_{i,k}}\left(O_i^{t_1}, \theta_{i,k}(O_i^{t_1}) + (q-1) * P(\tau_{i,k})\right) + 1\end{aligned} \tag{4.14}$$

Let $LO^{t_1}_{\text{SET}_p}$ be a list of alignments which consists of one $O_j^{t_1}$ for each transaction $Tr_j \in \text{SET}_p$. We can prove that $LO^{t_1}_{\text{SET}_p} \in \tilde{LO}_{\text{SET}_p}$.

Similar to the two analyses, an upper bound of BW_k is obtained by computing $\tilde{BW}^+_{LO^{t_1}_{\text{SET}_p}}$, the least fixed point of equation

$$\tilde{BW}^+_{LO^{t_1}_{\text{SET}_p}} = \tilde{f}_B^+(\tilde{BW}^+_{LO^{t_1}_{\text{SET}_p}})$$

Within $\tilde{BW}^+_{LO^{t_1}_{\text{SET}_p}}$, we can compute the maximum number $q^+_{\tilde{BW}^+_{LO^{t_1}_{\text{SET}_p}}}$ of activations of task $\tau_{i,k}$. Then, for each $q \leq q^+_{\tilde{BW}^+_{LO^{t_1}_{\text{SET}_p}}}$, we compute the least fixed point of equation:

$$\tilde{Q}^+_{LO^{t_1}_{\text{SET}_p}} = \tilde{f}_Q^+(q, \tilde{Q}^+_{LO^{t_1}_{\text{SET}_p}})$$

Consequently, the response time of jobs of task $\tau_{i,k}$ released in the busy window $[t_1, t_2[$ is upper bounded by

$$\tilde{RT}^+_{LO^{t_1}_{\text{SET}_p}}(\tau_{i,k}) = \max_{q \leq q^+_{\tilde{BW}^+_{LO^{t_1}_{\text{SET}_p}}}} \left\{ \tilde{Q}^+_{LO^{t_1}_{\text{SET}_p}}(q) - \underbrace{(\theta_{i,k}(O_i^{t_1}) + (q-1) * P(\tau_{i,k}))}_{\text{phase}} + C(\tau_{i,k}) - 1 \right\}$$

Finally, the WCRT $wcrt_{i,k}$ of task $\tau_{i,k}$ is the maximum of these values for all possible alignments represented by $\tilde{LO}_{\text{SET}_p}$.

Theorem 6.

$$wcrt_{i,k} \leq \max_{LO \in \tilde{LO}_{\text{SET}_p}} \tilde{RT}^{+}_{LO}(\tau_{i,k})$$

This analysis is a generic version for both the precise one and the approximate one. It provides with the same result as the precise analysis when SET_p only contains transaction Tr_i, while it returns the same result as the approximate analysis when $\text{SET}_a = \emptyset$. Of course, many other combinations can be used by using different divisions of transactions.

Its complexity is between the precise one and approximate one. In Theorem 6, the size of $\tilde{LO}_{\text{SET}_p}$ reflects the time complexity of this analysis. It is greater than the size of LO_i used for the approximate analysis and less than the size of $\{LO_1 \times \cdots \times LO_N\}$ used for the precise analysis. Its precision lies between the precise one and the approximate one. It is possible to obtain many different computations and precisions by choosing how to divide transactions into SET_a and SET_p.

4.4 Combined Analysis and Result Certifier

In this section, we present two combined RTAs based on the previous generic combined analysis. The main idea is to use the approximate version when it can be shown that its result is the precise one.

In addition, we would like to present CertiCAN, a result certifier combining several certified analyses, which can certify the results of industry analyzers for relatively large systems.

For the sake of simplicity of presentation, we start with a simple version of the full combined RTA. We call it a 2-level combined RTA, which combines two analyses (a precise analysis and an approximate analysis). Then, we present the full version, which is at the basis of CertiCAN.

4.4.1 2-level Combined RTA

The 2-level combined analysis is based on a precise and an approximate analyses. Its main features are:

- It uses the approximate analysis to avoid unnecessary computations and thus increase performance;
- It nevertheless computes the same results as the precise analysis.

The precise analysis, $\mathcal{A}_p$, considers a list of precise scenarios S_p and computes a worst case response time for each. Its result is the maximum of all these computations, that is:

$$\max_{s_p \in S_p} \mathcal{A}_p(s_p)$$

The approximate analysis, $\mathcal{A}_a$, does the same on a list of approximate scenarios S_a. The two key properties of a valid approximate analysis are that:

- Each approximate scenario s_a dominates a list of precise scenarios written Ds_a *i.e.*,

$$max_{s_p \in (Ds_a)} \mathcal{A}_p(s_p) \leq \mathcal{A}_a(s_a) \tag{4.15}$$

- The list S_a of approximate scenarios dominates all precise scenarios of S_p.

Therefore, we know that the result of the approximate analysis is an over approximation of the precise result, that is

$$max_{s_p \in S_p} \mathcal{A}_p(s_p) \leq max_{s_a \in S_a} \mathcal{A}_a(s_a) \tag{4.16}$$

The 2-level combined analysis is based on the following observation: If the WCRT obtained for an approximate scenario s_a is less than the WCRT found so far on the set of precise scenarios visited, then there is no need to analyze the precise scenarios dominated by s_a.

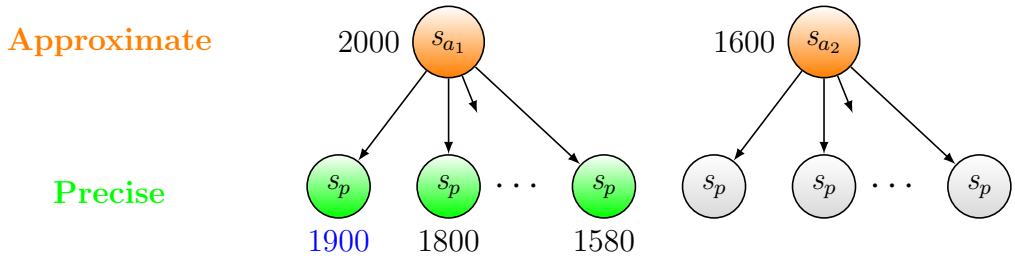


Figure 4.4: Scenario domination according to the 2-level combined analysis for a system of four transactions. Each approximate scenario (orange node) dominates 1000 precise scenarios (green nodes). Unprinted nodes and the nodes in gray represent the scenarios which do not need to be analyzed.

Example 2. *Consider a system of four transactions* $\{Tr_1, Tr_2, Tr_3, Tr_4\}$ *and the number of alignment candidates in each transaction is 2, 10, 10, 10 respectively. For a task of transaction* Tr_1*, the precise analysis has to perform 2000 scenarios for computing a precise result, while we only need to examine 2 scenarios for obtaining an approximate result by over-approximating the three other transactions* Tr_2*,* Tr_3*, and* Tr_4*.*

This example with 2 approximate scenarios and each one dominating 1000 precise scenarios is depicted in Figure 4.4, where vertices represent scenarios, edges

represent the domination relation and labels next to vertices are their corresponding response times. For this example, the 2-level combined analysis is called with an initial WCRT 0 *and the list* $[(2000, s_{a_1}); (1600, s_{a_2})]$. *It proceeds as follows:*

- *the current approximate response time (*2000*) is greater than the current WCRT (*0*) then the maximum response time of the 1000 dominated precise scenarios is computed (*1900*) and becomes the current WCRT;*
- *because* 1900 *is greater than or equal to the next approximate response in the list (*1600*), the response times of the corresponding dominated scenarios do not need to be computed (they are necessarily smaller);*
- *the analysis returns* 1900 *which is the precise WCRT.*

The 2-level combined RTA returns the same WCRT as the precise analysis but with fewer computations *i.e.,* it does not need to analyze the 1000 precise scenarios dominated by the second approximate scenario s_{a_2}.

Using this mechanism, we can derive a result certifier. Consider a R_0 computed by an industrial analyzer to be certified, we apply the approximate/precise analysis with an initial WCRT set at R_0.

4.4.2 Full Combined RTA

The problem of the 2-level combined RTA is that an approximate scenario may dominate a considerable number of precise scenarios, in particular, for systems that have many transactions. Therefore, it is often still intractable to compute a precise result for large systems. In order to solve this issue, we add intermediate approximate levels to get an n-level combined RTA. We refer to this version as the full combined RTA. This

The full combined analysis is based on the generic analysis as presented in Section 4.3.4, which is a generic version of both the precise analysis and the approximate analysis. The generic analysis separates all transactions into two disjoint sets SET_p and SET_a. It examines all precise scenarios for transactions in SET_p and takes an approximate workload bound function for each transaction in SET_a.

The main idea of the full combined analysis is to refine the approximate analysis one transaction by one transaction until finding the same result as the precise analysis. For instance, consider a system $\{Tr_1, \ldots, Tr_n\}$, to analyze a task in transaction Tr_1, the full combined analysis starts with the generic analysis with the setting $\text{SET}_p = \{Tr_1\}$ and $\text{SET}_a = \{Tr_2, \ldots, Tr_n\}$. The scenarios correspond to all combinations $\tilde{LO}_{\text{SET}_p}$ of precise alignments among transactions in SET_p, that is represented by LO_1; Then, if SET_a is not empty, we refine the result by putting one transaction

from SET_a into SET_p, and now $\text{SET}_p = \{Tr_1, Tr_2\}$ and $\text{SET}_a = \{Tr_3, \ldots, Tr_n\}$. Thus, we must now check more alignments $\tilde{LO}_{\text{SET}_p} = LO_1 \times LO_2$; Consequently, the time complexity increases but the result is more precise; When all transactions are moved into SET_p, the analysis checks all combinations of precise alignments. Therefore, it provides the same result as the precise analysis. Using this refinement technique, we build a tree with as many levels as the number of transactions in the system to analyze. Each level represents a setting of the generic analysis. The first level describes the approximate analysis, while the last level expresses the precise analysis.

Example 3. *We take the same example as in Example 2 and build a 4-level tree according to the procedure of the full combined analysis (in Figure 4.5). The full combined analysis first computes the sorted list* $[(2000, s_{a_1}); (1600, s_{a_2})]$ *representing scenarios at Level-1 and the transaction sets* $\text{SET}_p := \{Tr_1\}$ *and* $\text{SET}_a := \{Tr_2, Tr_3, Tr_4\}$ *to refine. Given that* SET_a *is not empty, we take one transaction from* SET_a *to* SET_p *and compute the results for the 10 scenarios dominated by the scenario corresponding to the biggest approximate response time (*2000*) at Level 1. Then the results for the 10 scenarios are sorted by descending order. That builds a part of Level-2; Recursively, we take one transaction from* SET_a *to compute Level-3, and so on until Level-4 when the* SET_a *is empty; At Level-4, all results are precise and the largest response time is* 1900*. Then,* 1900 *becomes the current precise WCRT; because* 1900 *is greater than or equal to all next approximate response at any level (*1890 *at Level 3,* 1700 *at Level 2,* 1600 *at Level 1), the response times of the corresponding dominated scenarios do not need to be computed (they are necessarily smaller); the analysis returns* 1900 *which is the precise WCRT.*

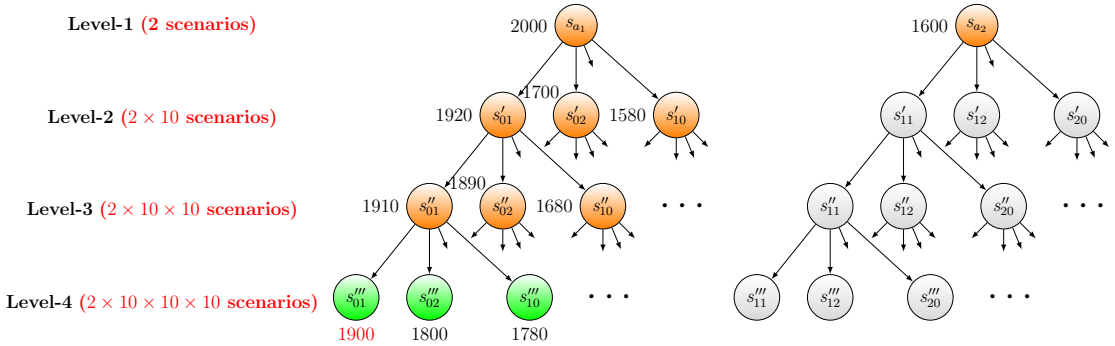


Figure 4.5: Scenario domination according to the full combined analysis for a system of four transactions. Each approximate scenario dominates 10 less approximate scenarios. Unprinted nodes and the nodes in gray represent the scenarios which do not need to be analyzed.

Consider the generic analysis $\mathcal{A}_g$ with a setting specified by $\text{SET}_p = \{Tr_1\}$ and $\text{SET}_a = \{Tr_2, Tr_3, Tr_4\}$ as the Level 1 in Example 3 and Figure 4.5. Its scenarios are denoted by $S_r = \{s_{a_1}, s_{a_2}\}$. The response time for a scenario $s \in S_r$ is $\mathcal{A}_g(s)$, *e.g.*, $\mathcal{A}_g(s_{a_1}) = 2000$ and $\mathcal{A}_g(s_{a_2}) = 1600$. The WCRT for this analysis is the maximum response time among all its scenarios S_r, that is $\max_{s \in S_r} \mathcal{A}_g(s) = 2000$. In order to get a more precise result, the refinement mechanism is applied if the set SET_a is not empty. It computes the scenarios dominated D_s by a scenario $s \in S_r$ *e.g.*, $D_{s_{a_1}} = \{s'_{01}, s'_{02}, \cdots, s'_{10}\}$. The result of this refinement depends on the selected transaction to refine from SET_a. We discuss the order in which transactions are chosen for the refinement in Section 4.5.3. After having applied the refinement technique, the two transaction sets are updated as $\text{SET}_p = \{Tr_1, Tr_2\}$ and $\text{SET}_a = \{Tr_3, Tr_4\}$, which it the setting for the new level *i.e.,* Level 2 in Figure 4.5. The scenarios at the new level are the concatenations D_{S_r} of all D_s such that $s \in S_r$. And the WCRT computed by the generic analysis with the new setting is $\max_{s' \in D_{S_r}} \mathcal{A}_g(s') = 1920$.

We can prove that the response time for any scenario in D_s is smaller than the one for its dominant scenario s, formally,

$$\max_{s' \in D_s} \mathcal{A}_g(s') \leq \mathcal{A}_g(s)$$

Further, the analysis with the new setting obtained using the refinement mechanism can find a more precise result than the previous setting, formally,

$$\max_{s' \in D_{S_r}} \mathcal{A}_g(s') \leq \max_{s \in S_r} \mathcal{A}_g(s)$$

Overall, the procedure of the full combined analysis is similar to the 2-level combined analysis: It uses the computed results from approximate scenarios to help the refinement procedure then to reduce the number of scenarios to analyze. Generally, it can be seen as a branch-and-bound algorithm applied to a tree.

The structure of the full combined analysis is shown in Algorithm 1. First, the generic analysis is first applied to each scenario at the first level (we denote S_1 the set of scenarios at the first level). These results (*i.e.,* one WCRT for each scenario) paired with their corresponding scenario are sorted in descending order $l_S := sort(map(\lambda s.(\mathcal{A}_g(s), s))\ S_1)$ as shown at Line 3 in Algorithm 1. Sorting the list in that order leads to considering the scenario with the largest approximate WCRT first. This heuristic relies on the intuition that the largest precise WCRT (which is the value to be found) is more likely to be dominated by a large approximate WCRT and therefore, will be found earlier with that ordering.

Algorithm 1 Combined RTA

```
1: % R denotes the current WCRT, its initial value is 0
2: % l_S denotes a list of scenarios (at the first level) paired with their corresponding
   WCRT
3: l_S := sort(map(λs.(A_g(s), s)) S_1)
4: % l_tr denotes a list of transactions (i.e., SET_a)
5: procedure CRTA(R, l_S, l_tr)
6:     match l_S with
7:     % l_S empty: returns the WCRT
8:     | nil ⇒ return R
9:     % otherwise: takes one element to analyze
10:    |(r, s) :: l'_S ⇒
11:    % if the current WCRT is greater than the approximate result r,
12:    then returns the current WCRT
13:    if R ≥ r then return R
14:    else
15:        match l_tr with
16:        % if l_tr is empty, returns max(r, R)
17:        | nil ⇒ return max(r, R)
18:        % otherwise, takes one transaction to refine
19:        |t :: l'_tr ⇒
20:        % computes the dominated scenarios as well as their results,
21:        and sorts their results
22:          l_local ← (sort (map (λs.(A_g(s), s)) (refine s l_tr)));
23:        % computes the result for those dominated scenarios
24:          R_local ← CRTA(R, l_local, l'_tr);
25:        % recursively, analyzes the remaining elements l'_S of the list l_S
26:          CRTA(R_local, l'_S, l_tr)
27:    end if
28: end procedure
29: CRTA(0, l_S, l_tr)
```

Then, at Line 5 in Algorithm 1, the combined RTA (CRTA) is called with 0 as the initial result (noted R, it will be updated after each iteration), the list $l_S := sort\ (map\ (\lambda s.(\mathcal{A}_g(s), s))\ S_1)$, and a list l_{tr} of transactions to be refined (that are transactions in SET_a). CRTA considers each approximate WCRT of list l_S in turn and starts with the first member (r, s) of l_S. Note that $r = \mathcal{A}_g(s)$ is the WCRT for scenario s. If $R \geq r$ then it stops and returns R (it is not the case when $R = 0$ as the initial input); Otherwise, at Line 15, examine whether there are still some transactions in l_{tr} to be refined. If l_{tr} is empty, that means that all transactions have been taken into account precisely and the current WCRT becomes $max(r, R)$; If there are still transactions in l_{tr}, for the current scenario s, at Line 22, the function *refine* compute its dominated scenarios as well as their results using the refinement mechanism and sort their results by descending order. These dominated scenarios paired with their results are stored in the list l_{local}. Next we calculate the local WCRT, noted R_{local}, for that list. Recursively, CRTA proceeds with the next element of the initial list and the local result R_{local} until it finds the global WCRT, *i.e.,* the precise WCRT. We have proven that the result returned by the combined analysis is the same as the one computed by the precise analysis.

4.4.3 Result Certifier

From the full combined RTA, we derive our result certifier, CertiCAN, which is able to check results of CAN analysis tools.

The algorithm of CertiCAN is shown in Algorithm 2. To check that R_0 is equal to or larger than the precise WCRT, CertiCAN considers each approximate WCRT of the argument list (*i.e., sort* $(map\ (\lambda s.(\mathcal{A}_g(s), s))\ S_1)$, noted by l_S) in turn and starts with the first member (r, s) of l_S. If the current WCRT is equal to or less than R_0 then the certification is completed (and returns *True*) since all remaining approximate WCRTs (and the WCRTs of the corresponding dominated scenarios) of the list l_S are also less than R_0. Otherwise, it examines whether there are still some transactions in l_{tr} to be refined. If l_{tr} is empty, it means that there is no transaction left to be refined and the current WCRT is a precise result, consequently if R_0 is smaller than that precise WCRT, CertiCAN returns *False*. Otherwise, for the scenario s, the function *refine* computes the dominated scenarios as well as their corresponding WCRT. These dominated scenarios paired with their corresponding WCRT are stored in the list l_{local}. Then, CertiCAN checks results computed using that list. If the result is *False* then the certification procedure completes and returns *False*. Otherwise, CertiCAN proceeds, recursively, with the next element of the initial list l_S.

Algorithm 2 The CertiCAN Result Certifier

R_0 contains the WCRT to certify

```
1: % l_S denotes a list of scenarios (at the first level) paired with their corresponding
   WCRT
2: l_S := sort(map(λs.(A_g(s), s)) S_1)
3: % l_tr denotes a list of transactions (i.e., SET_a)
4: procedure CERTICAN(l_S, l_tr)
5:     match l_S with
6:     % if l_S is empty, returns True
7:     | nil ⇒ return True
8:     % otherwise, takes one element of l_S to analyze
9:     |(r, s) :: l'_S ⇒
10:    % if the WCRT to certify is greater than the approximate result r,
11:    then returns True
12:    if R_0 ≥ r then return True
13:    else
14:        match l_tr with
15:        % if l_tr is empty, returns False
16:        | nil ⇒ False
17:        % otherwise, takes one transaction to refine
18:        |t :: l'_tr ⇒
19:        % computes the dominated scenarios as well as their results,
20:        and sorts their results
21:          l_local ← (sort (map (λs.(A_g(s), s)) (refine s l_tr)));
22:        % certifies the result for those dominated scenarios
23:          R_local ← CERTICAN(l_local, l'_tr);
24:        if R_local = False then return False
25:        else
26:            % recursively, certifies the results for the remaining elements l'_S
27:            CERTICAN(l'_S, l_tr)
28:        end if
29:    end if
30: end procedure
31: CertiCAN((sort (map (λs.(A_g(s), s)) S_1)), l_tr)
```

If CertiCAN returns *True* then it can be formally proved that R_0 is greater than or equal to the precise WCRT *i.e.,* the result produced by the precise analysis.

Our combined RTA and CertiCAN follow a principle that is similar to the abstraction refinement method used in [131, 132]. In particular, these two papers already use different abstraction levels to compute precise bounds with increased efficiency. The main difference is that [131] deals with the analysis of digraph tasks with constrained deadlines, which does not fit the CAN context. Also, this approach proves to be particularly well suited for result certification.

4.5 Optimization

In order for CertiCAN to certify results for large systems, we present three optimizations in this section. They are based on the following observations:

1. Within one transaction, there are possibly many alignments. Each alignment can be considered as a specific workload function to analyze as shown in Figure 4.6a. Therefore, we can determine the domination relation between alignments by comparing their corresponding workload functions. One can thus remove dominated alignment candidates without loss of precision;

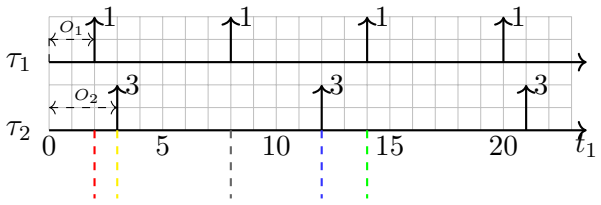
2. During the analysis, some functions are called with the same arguments repeatedly. This can be avoided using some techniques like lookup tables, memoization;

3. The order of transactions considered for refinement affects the analysis speed. We have investigated several heuristics to accelerate the analysis.

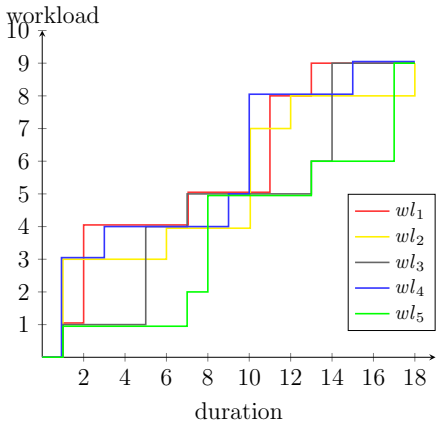
4.5.1 Removing dominated alignment candidates

Each alignment of one transaction corresponds to a workload function as shown in Figure 4.6. For instance, for the alignment represented by the red dashed line (at time 2) in Figure 4.6a, its cumulated workload for a given duration Δ is $wl_1(\Delta)$ in Figure 4.6b. Thus, a transaction can be considered as a set of workload functions to analyze. We design an algorithm that filters out dominated alignment candidates (*i.e.,* workload functions) from each transaction. For example, we can remove the function wl_5 because its value is always smaller than the one of wl_1. Then we prove that the filter is correct, that is, it preserves the final results.

First, let us introduce some definitions expressing relations between workload functions and between scenarios.



(a) A transaction of two periodic tasks. The LCM of tasks periods is 18, during which there will be 5 activations.



(b) Workload functions of the five alignment candidates of the transaction presented in Figure 4.6a.

Figure 4.6: Abstraction of one transaction to workload functions.

Definition 24 (Strong workload domination)**.** *A workload function wl_1 is said to strongly dominate the workload function wl_2, denoted $wl_1 \succeq wl_2$, if and only if for any duration $\Delta \in N$,*

$$wl_1(\Delta) \geq wl_2(\Delta)$$

Definition 25 (Weak workload domination)**.** *A workload function wl_1 is said to weakly dominate the workload function wl_2* w.r.t. *a duration L, denoted $wl_1 \underset{L}{\succeq} wl_2$, if and only if for any duration $\Delta \in [0, L]$,*

$$wl_1(\Delta) \geq wl_2(\Delta)$$

Note that to determine the weak domination relation between workload functions for a set of periodic tasks, it is sufficient to set L as the hyper-period of all task periods.

A scenario corresponding to a specific alignment can be considered as a collection of workload functions, which consist of one function from each transaction. We generalize the notion of domination to scenarios.

Definition 26 (Scenario workload)**.** *Consider a scenario s corresponding to a collection of n workload functions $\{wl_1, wl_2, \ldots, wl_n\}$, for a given time duration Δ, its workload is the sum of all workloads provided by the n workload functions. Formally,*

$$wl^s(\Delta) = \sum_{i=1}^{n} wl_i(\Delta)$$

Definition 27 (Scenario domination). *A scenario s_1 is said to dominate the scenario s_2, denoted $s_1 \succeq s_2$, if and only if the response time computed for s_1 is greater than the one obtained for s_2. Formally,*

$$s_1 \succeq s_2 \iff \tilde{RT}^+_{s_1} \geq \tilde{RT}^+_{s_2}$$

Lemma 14 (Workload function domination implies scenario domination). *Consider two scenarios s_1 and s_2, their workload function are wl^{s_1} and wl^{s_2}, respectively. Let L be the least fixed point of equation $f(L) = B_k + \sum_{Tr \in Sys} wl^*_{Tr}(L)$. If wl^{s_1} weakly dominates wl^{s_2}* w.r.t. *L, then $\tilde{RT}^+_{s_1} \geq \tilde{RT}^+_{s_2}$. As a result,*

$$wl^{s_1} \underset{L}{\succeq} wl^{s_2} \implies s_1 \succeq s_2 \tag{4.17}$$

In order to determine the domination relations between scenarios, we compute the weak domination relations between their corresponding workload functions. Then, we design a procedure to filter out dominated functions using domination relations and prove its correctness.

That algorithm relies on three simple functions:

1. *Filter*(l_f, f) removes the function f from the list l_f of functions if f is a member of that list;
2. *Compare*(wl_1, wl_2, L) computes whether wl_1 is weakly dominated by wl_2 *w.r.t. L*. It returns *true* if wl_1 is dominated by wl_2 for any $\Delta \in [0, L]$ and *false* otherwise;
3. *DominatedByOthers*(f, l_f, L), defined in Algorithm 3, computes whether a function f is weakly (*w.r.t. L*) dominated by any function except itself from a list l_f of functions.

The main procedure, presented in Algorithm 4, proceeds as follows:

- It starts with n workload functions $f_0, .., f_{n-1}$ to be filtered and a length L for computing their weak domination relations;
- DominantFunction is called to filter out all dominated functions from l_f (Line 15). It takes two lists (initially, they are the same *i.e.,* l_f) and a number L; it returns the list of all dominant functions of l_f (*i.e.,* the functions which are not dominated by any other function from l_f);
- if l_2 is empty then it returns the list l_1 (Line 4), otherwise it checks whether the first function wl from l_2 is dominated by any other function from l_1 (Line 8);

Algorithm 3 Dominated by Others

```
1: f denotes a workload function
2: l_f denotes a list of workload functions f_0, f_1, .., f_{n-1}
3: L denotes the sufficient length for determining the weak domination relations
   between workload functions from the list l_f
4: procedure DominatedByOthers(f, l_f, L)
5:     match l_f with
6:     | nil ⇒ false
7:     |f' :: l'_f ⇒
8:     % if f' is f itself (note that each function has an identifier)
9:     if f' is f then
10:        % then examine other functions l'_f
       return DominatedByOthers(f, l'_f, L)
11:        % else if f is dominated by f' return true
12:    else if Compare(f, f', L) then
       return true
13:    else
14:        % else examine other functions l'_f
       return DominatedByOthers(f, l'_f, L)
15:    end if
16: end procedure
```

- if wl is dominated by another function of l_1 then it removes wl from l_1 and continues to examine the functions l'_2 by calling DominantFunction (Line 9); otherwise,
- otherwise, it examines the remaining functions l'_2 without changing l_1 (Line 11).

As mentioned before, the LCM of task periods is one sufficient length for determining weak domination relation among workload functions. In order to increase the tool efficiency, we have proven the smallest sufficient length (SSL) for filtering out the dominated workload functions by computing the biggest busy window. We will compare the efficiency of the filter with LCM and the one with SSL in the next section. This optimization has been applied it in our tool.

4.5.2 Avoiding recomputations

To analyze the response time of one task, we need to examine many scenarios, which are actually combinations of workload functions. And to certify a system, we need to analyze all tasks. Workload functions will be evaluated numerous times including many recomputations. We can certainly improve this issue by applying techniques like lookup tables and memoization. For the sake of simplicity of proofs, we used

Algorithm 4 Dominant Function Filter

1: l_f denotes a list of workload functions $f_0, f_1, .., f_{n-1}$ to filter
2: L denotes the sufficient length for determining the weak domination relations between workload functions from the list l_f
3: **procedure** DOMINANTFUNCTION(l_1, l_2, L)
4: match l_2 with
5: | nil $\Rightarrow$ l_1
6: |$wl :: l'_2 \Rightarrow$
7: % if wl is dominated by any other function from l_1
8: **if** *DominatedByOthers*(wl, l_1, L) **then**
9: % then wl is removed from l_1 then the functions l'_2 are examined
return DOMINANTFUNCTION(*Filter*(l_1, wl), l'_2, L)
10: **else**
11: % else examine the functions l'_2
return DOMINANTFUNCTION(l_1, l'_2, L)
12: **end if**
13: **end procedure**
14: % l_f^{dom} contains all dominant functions for the list l_f
15: l_f^{dom} := DOMINANTFUNCTION(l_f, l_f, L)

lookup tables to store results of workload functions.

For each transaction, we can pre-calculate each workload function up to a large enough value[5], *e.g.,* its hyper period T^+ and store all values in a table. When a specific value is demanded, we search from this table, instead of recalculating. However, it is necessary to compute all workload values for the domain $[0, T^+[$. We only compute workloads for discontinued instants *i.e.,* when their values change. For instance, in Figure 4.6b, the instants when wl_1 must be computed are just 1, 2, 7, 11, 13, and similarly 1, 5, 7, 13, 14 for wl_2.

This optimization has been implemented and its correctness has been proven.

4.5.3 Heuristic algorithms

The analysis starts with approximate results, then it refines them by examining precise alignments one transaction by one transaction. The question is: which transaction should be selected first to analyze? We investigated whether the order in which transactions are precisely considered (*i.e.,* put in the set SET_p) affects the efficiency of the combined analysis and CertiCAN. For this, we implemented two different heuristics:

[5]It should be greater than the length of any busy window computed in analyses. In our implementation, it is the length of the largest busy window that is computed when analyzing the lowest task. We have proven its sufficiency.

- Static order. We sort transactions to refine by putting the transaction with the highest utilization first before performing the analysis. The intuition is that when computing an approximate workload, the higher the utilization of the transaction, the more pessimistic results probably are. In other words, if the transaction with the highest utilization is considered first for each refinement, the result is probably refined the most. Therefore, there will be more chances to speed up the analysis by using the utilization-sorted transaction list to refine. This configuration costs very little and works for most systems. This strategy is used by our tool. The gain of this optimization depends on the utilization distribution over transactions.
- Dynamic order. In the case where the utilization of the different transactions is similar, the static order does not provide significant benefit. In this case, we tried a dynamic order. For each refinement, we examine each transaction to refine (*i.e.,* compute the corresponding dominated scenarios as well as their WCRT) then choose the transaction with the largest WCRT to analyze. Thus, it needs quite a lot more computations to find the transaction at each step. In some cases, it speeds up the analysis, but slows it down for other cases. This optimization is not currently used in our tool.

4.6 Experimental Evaluation

Having completed the Coq formalization and correctness proofs of our analyses, we used the Coq extraction feature to obtain four certified tools: a precise analyzer[6], an approximate analyzer, a combined analyzer, and CertiCAN, the result certifier based on the combined analysis. Note that all these extracted analyzers implement the optimizations presented in Section 4.5.

In this section, we evaluate these certified tools in terms of performance and scalability.

The evaluated task sets are generated by NETCARBENCH[7], a benchmark generator for automotive message sets. This generator is used in the design and configuration of CAN and FlexRay communication systems. The following experimentations have been performed on 3000 systems that were generated by NETCARBENCH using a set of parameters presented in Section 4.1. More detail about configuration parameters[8] can be found in our NETCARBENCH configuration file in Appendix A.1.

[6]Given that the precise analyzer has a very high time complexity and can only deal with small systems, it is not considered in the following experimentations which focus on large systems.

[7]http://www.netcarbench.org/

[8]These parameters are provided by an expert from automotive domain. Note that the utilization

Table 4.1: Configuration parameters for NETCARBENCH generator.

ECUs	7 - 15
Utilization	40% - 60%
Period	$\{5, 10, 20, 50, 100, 200, 500, 1000\}$
Offset	random with granularity = 5
Priority	unique, arbitrary distribution
Transmission speed	500 kbits/s

In all figures, all results are obtained from an Intel Core i7@2.6GHz, 16Gb, 64bits laptop.

4.6.1 Evaluation of analyzers

First, we compare the three analyzers: the approximate analyzer, the combined analyzer, and RTaW-Pegase.

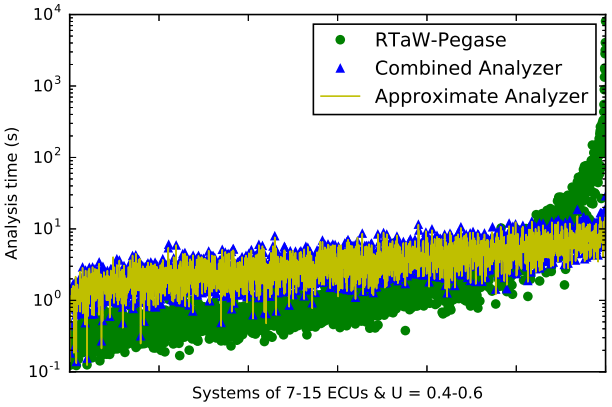


Figure 4.7: Comparison between certified analyzers and RTaW-Pegase

Figure 4.7 shows that the combined analyzer has a remarkable performance. It returns precise results and its time efficiency is close to the one of the approximate analyzer. Compared to Pegase, the combined analyzer has a better scalability. For the most complex systems, Pegase uses approximately two hours to compute a result whereas the combined analyzer needs less than 30 seconds to provide the same result. The main reason is that the combined analyzer combines two analyses (a

of the first ECU is allocated 30% of the whole utilization, *e.g.,* its 18% if the system utilization is 60%

precise and an approximate) in an optimized way that analyzes scenarios on demand. Other reasons may be that our optimizations avoid re-computations as much as possible, *e.g.,* by calculating the discontinued points and removing dominated scenarios. On the other hand, Pegase is more efficient than the combined analyzer on simple systems. One reason is that Pegase is written in C whereas the analyzer is extracted from Coq proofs in OCaml.

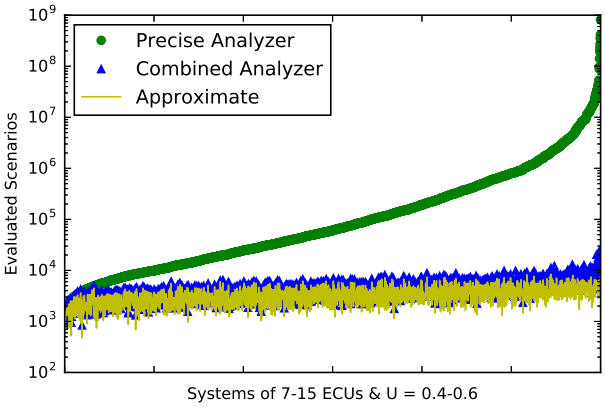


Figure 4.8: Evaluated scenarios by the certified analyzers

During our experimentations, we recorded the number of scenarios evaluated by the certified tools as showed in Figure 4.8. The number of scenarios for the precise analyzer is theoretically computed to be compared with the two other analyzers. This figure shows again that the combined analyzer has a good scalability. It is comparable to the approximate analyzer. Note that the trend of the combined and approximate analyzers' curves is similar to the one in Figure 4.7 because the runtime of the analyses depends directly on the number of scenarios that are evaluated.

4.6.2 CertiCAN *vs* Combined analyzer

We evaluate CertiCAN by verifying the results produced by the industrial tool RTaW-Pegase. We compared the performance of CertiCAN and the combined analyzer in Table 4.2.

The results in Table 4.2 show that CertiCAN is as good as the combined analyzer which is not surprising since they both share the same techniques and optimizations. Knowing the result to check, CertiCAN is a bit more efficient than the combined analyzer (by 17%). Both tools return a result in less than four seconds for most

Table 4.2: Performance comparison between CertiCAN and the combined analyzer.

	Tool	min	mean	median	max
# Evaluated scenarios	CertiCAN	496	3573	3330	17136
	Combined Analyzer	855	4392	4082	25126
Runtime (s)	CertiCAN	0.10	**3.67**	3.09	24.29
	Combined Analyzer	0.14	**3.85**	3.23	29.15

systems. For the most complex systems, they both only take less than half a minute. Note that the number of evaluated scenarios is not exactly proportional with the runtime because different scenarios may have a different time complexity, *e.g.,* an approximate scenario requires more computations to find the maximum workload among all its workload functions.

In addition, we evaluated 100 more complex systems with the configuration (utilization = 60 - 80 %, 15 - 20 ECUs) presented in Appendix A.2. In this experiment, we verify the system schedulability, *i.e.,* using task deadlines as CertiCAN inputs. With this setting, CertiCAN is 45 times more efficient than the combined analyzer.

According to all our experiments, we found that both the combined analyzer and CertiCAN have a high scalability. As far as we know, in modern cars, no more than 15-20 ECUs are connected to a single CAN bus [107, 113]. This indicates that CertiCAN can provide formal guarantees for industrial CAN bus analyzers.

4.6.3 Impact of optimizations

Additionally, we have generated 100 systems using the same configuration presented in Table 4.2 then evaluated them to understand the impact of optimizations. The result is presented in Table 4.3. CertiCAN with 2 levels cannot deal with large systems, using many levels of refinement and a filter for filtering out dominated functions makes it possible. The filter with SSL (smallest sufficient length) is 5 - 8 times more efficient than the one with LCM. The optimization of avoiding recomputations[9] contributes a factor of 15 - 25.

Table 4.3: Impact of optimizations.

	Statistic	**2 levels**	Many Levels Filter-LCM Without AVO	Many Levels Filter-SSL Without AVO	Many Levels Filter-LCM With AVO	Many Levels Filter-SSL With AVO
CertiCAN runtime (s)	Mean	-	492	61	20	**4**
	Max	-	6435	329	831	**15**

[9]In Table 4.3, AVO stands for avoiding recomputations, *i.e.,* lookup table, discontinued points.

4.7 Discussion

4.7.1 Experience with the Coq proof assistant

We have formally proven in Coq the correctness of the precise, approximate and generic RTAs presented in Section 4.3. Our proofs build upon the Prosa library [112] and use the basic definitions that it provides (task, job, arrival sequence, schedule, busy window, etc.). The combined analyzer and the result certifier (see Section 4.4.3) as well as their optimizations have also been specified and proved correct in Coq. Note that if proofs are machine-checked, this cannot be the case for specifications and theorem declarations. Besides using some basic definitions from Prosa, we also had to define the FPNP scheduling policy and the task model. Those specifications are quite small and simple compared to the proofs and can be scrutinized by the interested reader [32].

Table 4.4 illustrates the complexity of the proof effort (note that it excludes the proof from Prosa).

Table 4.4: Proof effort for certifying CAN analyses.

Feature	**LOC**
System model (with proof)	1000
Workload property & removing re-computation	3142
Fixed point property	700
Busy window analysis	3037
Generic analysis	294
Combined analysis	1680
Combination property	545
Approximate and precise analyses	432
Candidate property	1562
Removing dominated candidates	1060
Analyzers & CertiCAN (with proof)	4000
Arithmetic proofs	1400
Total	18852

Formalizing these developments in Coq requires more time and effort than on paper, but it also brings important benefits:

- It gives formal guarantees about the soundness of the specification and the absence of flaws in the proofs;
- It provides a better understanding of the role of each assumption, which helps to generalize proofs;

- The Coq extraction technique permits to produce formally verified tools (such as analyzers and certifiers) in the form of OCaml programs.

4.7.2 Possible extensions of the approach

One of the most interesting by-products is that formalization often leads to more general and reusable proofs. For instance, our proof of busy window analysis does not rely on a specific task model but on abstract functions. It can be reused for other task models as long as we have the corresponding abstract functions.

Also, we defined two RTAs by instantiating the abstract functions with two different workload functions. Actually, the approach could be applied to get other RTAs with different levels of approximation. The combined analysis and certifier depend of generic properties (domination relations) that do not rely on the specific real-time model under study. The correctness proofs for the combined algorithm would apply to many other kinds of analyses possibly disconnected from real-time theories.

4.8 Conclusion

In this chapter, we have presented CertiCAN, a tool extracted from Coq proofs for the certification of CAN analysis results.

The analysis underlying CertiCAN is based on a combined use of two well-known CAN analysis techniques, one precise and the other approximate. The resulting analysis is as tight as the precise analysis, but much faster. All three analyses have been proven correct in Coq on top of the Prosa library.

We have shown that CertiCAN is efficient enough in terms of computation time, the CertiCAN approach, which provides result certification rather than tool certification, is a realistic solution for industry practice. The reason for this is twofold: First, it is flexible and light-weight in the sense that it does not depend on the internal structure of the analysis tool that it complements. Second, it is efficient enough in terms of computation time. In particular, it is able to certify results computed by RTaW-Pegase, an industrial CAN analysis tool, even for large systems. This work suggests RTaW-Pegase implement all the optimizations proven in this chapter to make it more efficient.

We believe that this work represents a significant step toward a formal certification of real-time systems analysis results in general. In particular, the underlying technique can be reused for any other system model for which there exist RTAs with different levels of precision.

Chapter 5

Generalized Digraph Model

Applying theorem proving in the formal verification of RTAs requires a large proof effort. It is time consuming to prove one RTA. To deal with this issue, we propose a very expressive task model and prove its corresponding RTA, then many more specific (standard or novel) analyses boil down to specifying and proving their translations into that model.

In this chapter, we propose a task model that generalizes the digraph model and its corresponding analysis for fixed-priority scheduling with limited preemption. A task may consist of several types of jobs (also called sub-tasks), each with its own WCET, priority, non-preemptable segments and maximum jitter. With such parameters, it is expressive enough to model intra- and inter- tasks dependencies. It encompasses many task model such as periodic/sporadic taks, arrival curves, digraph tasks, transactions with offsets and so on. We present the correctness proof of the analysis in a way amenable to its formalization in the Coq proof assistant.

Contents

5.1 Motivation and Objective

Formal verification of real-time analyses with the help of a proof assistant like Coq has become an active field of research in the real-time community because it provides high confidence in the correctness of the analyses. However, it requires an important human effort, so making proofs general, generic, and/or reusable is of great importance.

Our objective is to propose a very expressive task model and to prove its corresponding RTA, such that many more specific (standard or novel) analyses boil down to specifying and proving their translations into that model. There exists a wide variety of task models and analyses for such policies. However, most of these models are incomparable and very few can describe both intra- and inter- task dependencies. The DRT task model [129] seems a good candidate for modeling intra-task dependencies, but its ability to capture inter-task dependencies is very limited. It cannot, for example, capture Tindell's offset model [142] presented in Section 2.1.

In this chapter, we propose a task model that generalizes the DRT model and its corresponding RTA, with the restriction that we consider only discrete time while some results about DRT apply also to dense time. A task may consist of several types of jobs (also known as sub-tasks), each with its own WCET, priority, non-preemptable segments (*i.e.,* its WCET is separated into several execution segments, it cannot be preempted within a segment) and jitter. Our model can capture dependencies between jobs of the same task as well as jobs of different tasks. We focus on fixed-priority scheduling policies and our model can encompass preemptive and non-preemptive models, as well as limited preemption. Despite being much more general, the RTA for our model is not significantly more complex than the original one. Also, it underlines similarities between existing analyses, in particular the analysis for the DRT model and Tindell's offset model.

We present the correctness proof of the analysis in a way amenable to its formalization in the Coq proof assistant and in the Prosa library. For the time being, the proof of the RTA of the general model within the Coq proof assistant is not yet complete. When it is certified, obtaining a certified more specific RTA will boil down to specifying and proving its translation into our model. Note that it may need more work to get an efficient RTA analyzer. Furthermore, expressing many different RTAs in a common framework paves the way for formal comparisons and generalizations (*e.g.,* design of novel RTAs).

The main contributions are:

- A general task model which encompasses complex dependencies between jobs and tasks;
- A RTA for that model;
- A correctness proof of that RTA amenable to its formalization in Coq and applicable to other task models.

5.2 System Behavior

In order to make the work amenable to its formalization in Coq, more precisely in Prosa, we begin with specifying system behaviors. We target concrete systems implemented as a set of tasks executing on a uniprocessor. The execution proceeds according to a *job-level fixed-priority limited preemptive* (JFPLP) scheduling policy. A JFPLP scheduler arbitrates between jobs competing for processor time by choosing the highest priority job, but it can only preempt a running job at some predefined execution points. In other words, each job is decomposed into non-preemptable

segments. This model subsumes the FPP and FPNP policies, and permits mixed policies [1]. A task may consist of different types of jobs (also called sub-tasks). Jobs of the same type have similar properties, in particular they have the same priority.

5.2.1 Definition of system behavior

We assume a set $\mathbb{T}$ of type names (types, for short). Each type entails a number of characteristics that are described in Section 5.3.

Let us recall the job definition specified in Section 2.4, a job is an instance of a task with a positive cost. We now generalize and enrich this definition using some functions in order to make it more expressive.

Definition 28 (Enriched Job)**.** *A job $\jmath$ is specified by:*

- *its* type $\mathbf{v}(\jmath) \in \mathbb{T}$*;*
- *its* priority $\mathbf{k}(\jmath) \in \mathbb{N}$ *inherited from its type; A greater number means a higher priority.*
- *its* arrival time $\mathbf{a}(\jmath) \in \mathbb{N}$*;*
- *its* jitter $\mathbf{j}(\jmath) \in \mathbb{N}$ *(also called release delay);*
- *a vector* $\vec{\mathbf{c}}(\jmath) = \langle c_1, \ldots, c_s \rangle$*,* $c_i \in \mathbb{N}^+$*, of durations corresponding to the* cost *(*i.e.*, execution time) of each non-preemptable segment.*

The cost *(or required service time) of a job $\jmath$ as above is* $\mathbf{c}(\jmath) = \sum_{1 \leq i \leq s} c_i$*. The* release time *of $\jmath$ is* $\mathbf{r}(\jmath) := \mathbf{a}(\jmath) + \mathbf{j}(\jmath)$*.*

We do not exclude different jobs from having the same parameters, but we assume that they can be distinguished (*e.g.,* through an identifier). We also assume that the set of jobs is partitioned into *tasks*. The behavior of a system is described using a set of executions defined by two infinite traces as used in Prosa: a *job arrival sequence* and a *schedule* presented in Section 2.4. The following schedule is a uniprocessor schedule.

Definition 29 (Job arrival sequence)**.** *A* job arrival sequence *is a function ρ mapping any time instant t to a finite (possibly empty) set of jobs $\rho(t)$ such that $\jmath \in \rho(t)$ iff $\mathbf{a}(\jmath) = t$.*

Definition 30 (Schedule)**.** *A* schedule *is a partial function σ which maps any time instant t to the job (if any) that is scheduled (*i.e.*, receives service) at t. A job $\jmath$ can be scheduled only when it is pending* i.e.*, it is released but not finished.*

Definition 31 (JFPLP schedule)**.** *A* JFPLP schedule *is a schedule such that the job that is scheduled is: either the job that is already executing one of its non-preemptable*

segments, or a job that has the highest priority h among pending jobs. If there are several pending jobs with priority h, a task is arbitrarily selected among those with such jobs; the chosen job is then the first released job with priority h in this task (FIFO policy).

Note that once the set of jobs and the function arrival time $\mathbf{a}$ are given, ρ is uniquely determined. A job $\jmath$ *completes* when it has received as much service time as it required, which is determined by the schedule. To recall some notations, we denote its completion time by $\mathbf{end}(\jmath)$. The *response time* of $\jmath$ is defined as $R_\jmath := \mathbf{end}(\jmath) - \mathbf{a}(\jmath)$. From its release time and until completion, a job is said to be *pending*.

5.2.2 Additional definitions and notations

In the following, we introduce additional definitions and notations needed in the following section.

Definition 32 (Job release sequence)**.** *Let ρ be a job arrival sequence. The corresponding job release sequence, written $\hat{\rho}$, is defined as $\hat{\rho}(t) := \{\jmath \mid \exists t', \jmath \in \rho(t') \wedge t = t' + \mathbf{j}(\jmath)\}$. We have $\jmath \in \hat{\rho}(t)$ iff $\mathbf{r}(\jmath) = t$.*

That is, the released sequence is the arrival sequence plus its jitters. For instance, $[(t_1, \{\jmath_1, \jmath_2\}), (t_2, \{\jmath_3\}), \ldots]$ denotes a sequence ρ such that $\rho(t_1) = \{\jmath_1, \jmath_2\}$, $\rho(t_2) = \{\jmath_3\}$ and $\rho(t) = \emptyset$ for all instants t absent from the list.

The *restriction* of a job arrival sequence ρ to a time interval $[t_1, t_2[$ is denoted $\rho/_{[t_1,t_2[}$. The same applies to job release sequences.

For a given set of jobs and a time duration, we can compute its requested workload by a job arrival sequence and its received service by a schedule.

Definition 33 (workload)**.** *Let ρ be a job arrival sequence and V a set of job types. The* workload *$wl_{V,\rho}$ of jobs with type in V in a time interval $[t, t+\Delta[$ is the cumulative cost (*i.e., *required service time) of such jobs released in that interval. Formally,*

$$wl_{V,\rho}(t, \Delta) := \sum_{\substack{\jmath : v \in V \\ t \le \mathbf{r}(\jmath) < t+\Delta}} \mathbf{c}(\jmath) \tag{5.1}$$

Definition 34 (Service time)**.** *Let σ be a schedule and V a set of job types. The* service time *$serv_{V,\sigma}$ received by jobs with type in V in a time interval $[t_1, t_1 + \Delta[$ is*

$$serv_{V,\sigma}(t_1, \Delta) := \sum_{\substack{t \in [t_1, t_1+\Delta[\\ \mathbf{v}(\sigma(t)) \in V}} 1 \tag{5.2}$$

The two definitions of workload and service time are used to perform a generic analysis using the concept of busy window which has been formally specified in Section 4.2 and which we recall here. The following definitions are implicitly parameterized by a job arrival sequence ρ and a schedule σ.

Definition 35 (Level-k quiet time)**.** *An instant t is said to be a* level-k quiet time *if all jobs of priority higher than or equal to k released strictly before t have completed at t.*

Definition 36 (Level-k busy window)**.** *A time interval $[t_1, t_2[$ is said to be a* level-k busy window *if:*

1. *t_1 and t_2 are level-k quiet times;*
2. *there is no level-k quiet time in $]t_1, t_2[$; and*
3. *at least one job with a priority higher than or equal to k is released in $[t_1, t_2[$.*

The last condition excludes degenerate cases of busy windows in which no job is scheduled. Since several jobs with the same type may be released in the same busy window, an additional concept of queueing prefix is required and has been presented in Chapter 4. We adapt it to task models considering non-preemptable segments.

Definition 37 (Queueing prefix)**.** *The q-th* queueing prefix of jobs of type v *in a level-k busy window $[t_1, t_2[$ is the time interval $[t_1, t_q]$ where t_q is the instant at which the last non-preemptable segment of the q-th job of type v receives its first service (*i.e., *is scheduled for the first time).*

Example 4. *Figure 5.1 presents a job arrival sequence ρ (split into the two* task-level *sequences ρ_1 and ρ_2) in a system made of two tasks. One task produces jobs of type v, priority 1, jitter 0 and segments $\langle 2, 2\rangle$ and two consecutive arrival times of its jobs are separated by at least 9 time units. The other task produces jobs of priority 2. The three first queueing prefixes of jobs of type v in the level-1 busy window $[0, 27[$ are $Q_{v,\rho}(1)$, $Q_{v,\rho}(2)$ and $Q_{v,\rho}(3)$.*

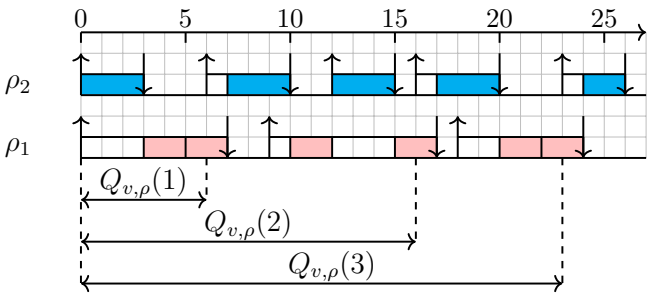


Figure 5.1: Queueing prefixes of jobs of type v.

5.3 Generalized Digraph Model

Our task model, called the *generalized digraph* (GD) model, is an extension of the digraph model [129] (presented in Section 2.1) with job level priorities, possibly null inter-arrival times, jitter and non-preemptable segments.

5.3.1 Syntax

A system consists of a set of n independent tasks $\Sigma := \{G_1, \ldots, G_n\}$, each task being specified by a graph $G_i := (V_i, E_i)$ where:

- V_i is a set of vertices representing different job types;
- E_i is a set of edges such that an edge connecting two vertices v_1 and v_2 of V_i is labeled with a duration $d(v_1, v_2) \in \mathbb{N}$ representing the minimum inter-arrival time between jobs of types v_1 and v_2.

A job type v is characterized by the following parameters:

- $\mathbf{K}(v) \in \mathbb{N}$ defines the priority of jobs of type v;
- $\mathbf{J}(v) \in \mathbb{N}$ specifies the maximum jitter (*i.e.,* delay between arrival and release time) for jobs of type v;
- $\vec{\mathbf{C}}(v) = \langle C_1, \ldots, C_s \rangle$ is a vector specifying the maximum cost of each non-preemptable segment of jobs of type v; $\mathbf{C}(v) = \sum_{i=1}^{s} C_i$ defines the maximum cost of jobs of type v.

Note that jobs have arbitrary deadlines and the RTA presented in this chapter does not rely on this parameter.

The sets of vertices V_i are assumed to be disjoint (*i.e.,* tasks activate jobs of different types). The set of vertices of the complete system is denoted by $V_\Sigma = \bigcup_{i=1}^{n} V_i$. For simplicity and to improve readability, we assume in this chapter that the jitter is *constrained*[1]: the jitter of any vertex $v \in V_\Sigma$ is smaller than or equal to the minimum inter-arrival time labeled on any edge going out of v.

In contrast with the standard digraph task model, null inter-arrival times are allowed. We however disallow tasks (*i.e.,* graphs) that contain null cycles (which would permit an infinite nomber of job arrivals at the same instant). This constraint is easily verified statically.

Example 5. *The graph G_e represented in Figure 5.2 defines a task with three types of jobs (*i.e., *three vertices). Vertex u is decorated with the triplet $(\langle 1, 1\rangle, 0, 1)$ where $\langle 1, 1\rangle$ indicates that jobs of this type can be preempted at each instant (segments of length 1) and that their maximum cost is 2(that is, $1 + 1$); their maximum release*

[1]The extension to arbitrary jitter is discussed in Section 5.4.3.

jitter is 0 *and their priority is* 1*. Similarly, jobs of type* v *have a maximum cost of* 5 *and cannot be preempted, their maximum jitter and priority are* 3 *and* 2*, respectively. Jobs of type* w *have a maximum cost of* 8 *and can be preempted once after their first segment (it could be 1, 2, 3, or 4) is completed.*

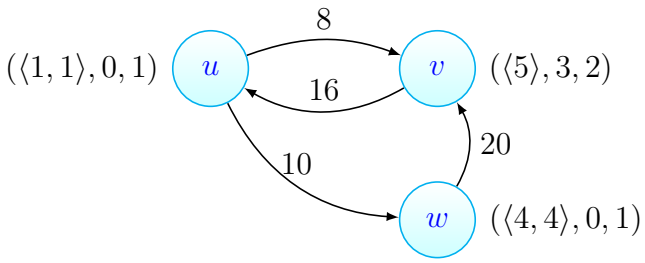


Figure 5.2: A graph G_e specifying a task with 3 types of jobs

5.3.2 Task-level and system paths

As a graph, a task $G_i = (V_i, E_i)$ specifies a set of paths, i.e., sequences of vertices of G_i such that $(v_j, v_{j+1}) \in E_i$. Note that a task will typically have cycles, thus specifying an infinite set of paths, some of them being infinite as well. In the following, we sometimes refer to paths of tasks as task-level paths for clarity.

Example 6. *In Figure 5.2,* $[u, v, u, w, v]$*,* $[w, v, u, v, u]$*, and the infinite sequence* $X = [u, v, X]$ *are paths of* G_e*, but* $[u, v, w]$ *is not.*

Definition 38 (System path)**.** *A* system path *of system* Σ *is a set* $\pi := \{\pi_1, \ldots, \pi_n\}$ *such that for each* i*,* π_i *is a task-level path of task* G_i *in* Σ*. The set of system paths of* Σ *is denoted* Π_Σ*.*

We use the following functions on task-level paths.

- The function *len* returns the sum of all minimum inter-arrival times on the edges of a finite path. Formally:

$$len([v_1, v_2, \ldots, v_k]) := \sum_{1 \le j < k} d(v_j, v_{j+1})$$

- The function $pre_\Delta(\pi_i)$ returns the longest prefix π_p of π_i such that $len(\pi_p) < \Delta$. A variant, written $pre^n_v(\pi_i)$, returns the prefix of π_i up to the n-th occurrence of vertex v in π_i.
- We write $len^n_v(\pi_i)$ for $len(pre^n_v(\pi_i))$ which returns the sum of all minimum inter-arrival times between the first vertex and the n-th occurrence of vertex v in π_i.

- The function $cost(seq)$ returns the sum of the maximum costs of all vertices in a vertex sequence seq.
- The filter function $|\pi|_V$ returns the vertex sequence obtained from π where all vertices not belonging to V have been filtered out.

5.3.3 Semantics

The semantics of a system $\Sigma := \{G_1, \ldots, G_n\}$ is given by the set of arrival sequences that are *consistent* with a system path of Σ. Consistency between an arrival sequence and a system path ensures that jobs in the sequence satisfy the constraints imposed on them by their type. The order and timing of job arrivals is compatible with the constraints specified on the edges of the graphs G_i.

Definition 39 (Consistency of an arrival sequence *w.r.t.* a path)**.** *An arrival sequence ρ is* consistent *with a task-level path $\pi_i = [v_1, v_2, \ldots]$, which is denoted $\rho \sim \pi$, iff there exists a flattening*[2] *$[(t_1, \jmath_1), (t_2, \jmath_2), \ldots]$ of ρ such that for all k:*

- *$\jmath_k$ is consistent with v_k,* i.e., *:*
  - $\mathbf{v}(\jmath_k) = v_k$*;*
  - $\mathbf{k}(\jmath_k) = \mathbf{K}(v)$*;*
  - $\mathbf{j}(\jmath_k) \leq \mathbf{J}(v_k)$*; and*
  - $\vec{\mathbf{c}}(\jmath_k) = \langle c_1, \ldots, c_s \rangle \wedge \vec{\mathbf{C}}(v_k) = \langle C_1, \ldots, C_s \rangle \wedge c_i \leq C_i, i = 1 \ldots s.$
- $d(v_k, v_{k+1}) \leq t_{k+1} - t_k$

The definition naturally extends to system-level paths.

We write $\rho \sim \Sigma$ to denote that an arrival sequence ρ is consistent with a system path in Π_Σ.

5.4 Expressiveness

In this section, we show how a variety of existing task models can be expressed using the GD model. We also hint at extended or new models that could be defined as GD and analyzed by the proposed RTA proposed in Section 5.5.

[2]For example, the arrival sequence $[(t_1, \{\jmath_1, \jmath_2\}), (t_3, \{\jmath_3\})]$ can be flattened into either $[(t_1, \jmath_1), (t_1, \jmath_2), (t_3, \jmath_3)]$ or $[(t_1, \jmath_2), (t_1, \jmath_1), (t_3, \jmath_3)]$.

5.4.1 Models without task dependencies

The GD model can easily emulate many kinds of arrival models. Obviously, a simple sporadic task whose jobs are of type v and with a minimum inter-arrival time p is represented by a single vertex v and self-loop labeled with p. A periodic task of period p can be represented by the same GD task. Of course, this GD task represents many more consistent job arrival sequences but the worst case analyzed by the RTA is precisely the periodic sequence.

Arrival curves [141] represent more expressive arrival models. For instance, the minimal distance function $d^-(a)$ returns the smallest time interval that may contain $a+1$ occurrences of a job. This function must be super-additive *i.e.,* $d^-(a)+d^-(b) \leq d^-(a+b)$. In general, an arrival curve may have an infinite description in terms of GD task (*e.g.,* $d^-(a) = a^2$ is super-additive and describes ever growing inter-arrival times). However, such functions[3] are usually given by a collection of values from 1 to some constant k. Then, the analysis uses the minimal super-additive extension (*e.g.,* the smallest function compatible with $d^-(1), \ldots, d^-(k)$). Such super-additive closures can be represented faithfully by a GD task. When d^- is convex (*i.e.,* $d^-(x+1) + d^-(x-1) - 2d^-(x) \geq 0$) then the minimal super-additive extension is periodic and $\forall x = qk + r, d^-(x) = qd^-(k) + d^-(r)$. The corresponding GD task is then made of a cycle of k vertices $v_0, \ldots, v_{k-1}$ where the minimum inter-arrival time decorating each edge is $d(v_i, v_{(i+1) \bmod k}) = d^-((i+1) \bmod k) - d^-(i)$.

Example 7. *Consider, for instance, the arrival curve specified by* $d^-(1) = 2, d^-(2) = 5, d^-(3) = 10, d^-(4) = 20$ *which specifies that* 2 *(resp.* 3*,* 4*, and* 5*) jobs cannot arrive in less than* 2 *(resp.* 5*,* 10 *and* 20*) time units. The corresponding* GD *task is given in Figure 5.3 (a).*

For non convex functions, it has been shown that their minimal super-additive extensions are pseudo-periodic functions [22] which can also be represented by a finite GD task.

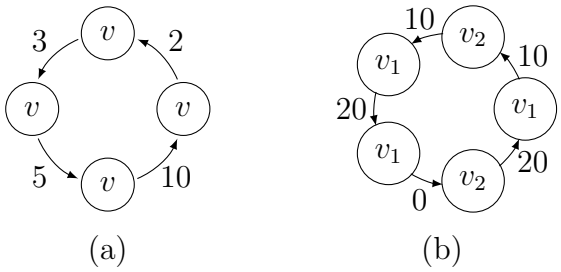


Figure 5.3: GD representing (a) an arrival curve model and (b) a transaction with offsets *a la* Tindell)

[3] By default $d^-(0) = 0$

5.4.2 Models with job and task dependencies

As a generalization of digraphs, the GD model can of course express all task models with intra-task dependencies that standard digraphs can. Let us cite, the multiframe model [105] and its generalized version [14], the recurring branching [15] or recurring RT [16] models, the non cyclic RT [13] and digraph model (DRT) [129].

Allowing job-level fixed priorities and null minimal inter-arrival times allows the GD model to model inter-task dependencies (*e.g.,* fixed timing relation among tasks, the offset model of Tindell [142]) which cannot be represented in the standard DRT model.

In Tindell's model (used in Chapter 4), a system is made of a set of independent transactions $\{Tr_1, \ldots, Tr_n\}$. Each transaction Tr_i consists of a set of periodic tasks with offsets $Tr_i := \{\ldots, \tau_{i,k}, \ldots\}$ where each task $\tau_{i,k}$ has its own WCET, period, offset, priority, and deadline. A transaction and its set of periodic tasks with offsets can be represented within a single GD task. It is sufficient to compute the hyper-period of the transaction and to build the circular GD task representing the inter-arrival time (which may be null) between arrivals of the different jobs in the hyper-period. The periods and offsets are represented by inter-arrival times. The WCET, priority and deadline of tasks are represented by the corresponding vertices.

Example 8. *Consider, for instance, a transaction Tr with two periodic tasks with jobs of type* v_1 *and* v_2*, with periods* 20 *and* 30 *and with offsets* 5 *and* 15*. The hyper-period is* 60 *and taking the first arrival of* v_1 *as the time origin, the arrival times are* $[(v_1, 0), (v_2, 10), (v_1, 20), (v_1, 40), (v_2, 40), (v_1, 60)]$*. The transaction Tr is represented by the* GD *task in Figure 5.3 (b). Its worst job arrival sequence* w.r.t. *the RTA is exactly the job arrival sequences of Tr.*

Shared resources, which entail inter-task dependencies, is a common issue in hard real-time systems. Abdullah et al. [1] addressed this problem using DRT by allowing two kinds of vertices : preemptable (tasks) and non-preemptable (resources). They proposed an extension of the RTA to take into account these two kinds of vertices. Using the GD model, this is directly modeled using segments with always preemptable vertices for tasks and non-preemptable ones for resources.

Rendez-vous mechanisms, another kind of inter-task dependencies, have been expressed using an extension of digraphs (SDRT) [104]. We believe that the encoding of such inter-task synchronization is possible in GD tasks but the exact encoding as well as its complexity remain to be investigated.

5.4.3 Beyond existing models

All these features, including jitter, can be combined to express and analyse new task models. For instance, non-preemptable segments and jitter have not been considered by many intra-task dependencies task models (*e.g.,* MF, GMF, RB, RR, DRT) nor does Tindell's model consider intra-task dependencies (*e.g.,* if a transaction could be a set of MF tasks instead of simple periodic tasks). Since all these models can be expressed as the GD model, they can be analyzed using the RTA described next.

Note that our RTA targets the GD model with constrained jitters. Any GD task with arbitrary jitters can be transformed into a GD task with constrained jitters with the same worst case scenario for the RTA. For instance, a sporadic task of the minimum inter-arrival time 10 and jitter 22 represented by G can be transformed into G' with jitter 0 in Figure 5.4. For a given time duration, the maximum cumulated workloads of the two tasks (original and transformed) are the same.

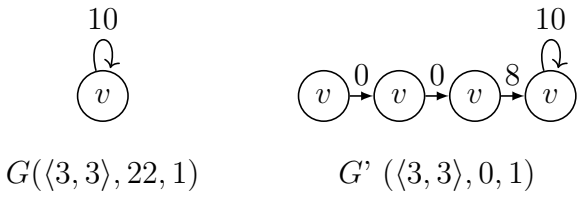


Figure 5.4: Encoding of arbitrary jitter

5.5 Response Time Analysis

The objective of RTA is to bound, as tightly as possible, the *worst-case response time* of each vertex (job type) v, defined as the maximum response time among all jobs of type v occurring in all arrival sequences and schedules consistent with each possible system path. Formally,

$$wcrt(v) := \max\{R_\jmath \mid \jmath : v \wedge \exists \pi \in \Pi_\Sigma, \exists \rho \sim \pi, \jmath \in \rho\}$$

In this section, we present the overall structure of the analysis that we propose for GD systems. We suppose given a vertex v, with priority k, of a task $G_i \in \Sigma$ and focus on upper bounding its worst-case response time.

Our analysis relies on a path-specific analysis of level-k busy windows, hence the following definition (implicitly parameterized as before by a job arrival sequence ρ).

Definition 40 (Busy window path)**.** *A level-k busy window* $[t_1, t_2[$ *is said to be* represented by *the shortest system path* π *such that the corresponding job release sequence* $\hat{\rho}/_{[t_1,t_2[} \sim \pi$*, and any extension of it.*

The shortest path representing a busy window $[t_1, t_2[$ is the path obtained after restricting the job release sequence to $[t_1, t_2[$. In other words, a busy window path is an abstraction of a level-k busy window. Equipped with these notions, the general principle of our RTA analysis can now be presented.

5.5.1 Overall structure of the RTA of Gd systems

The RTA of a vertex v with priority p, of task G_i, consists in analyzing a set of paths such that all possible level-p busy windows are represented by one path in the set. The methodology to bound the worst-case response time of jobs of type v is thus as follows.

Step 1: Derive a set of system paths $\mathbf{\Pi}^v_\Sigma$ such that any possible level-p busy window (for any arrival sequence and schedule consistent with any system path in Π_Σ) is represented by one path in $\mathbf{\Pi}^v_\Sigma$.

Step 2: For each system path $\pi \in \mathbf{\Pi}^v_\Sigma$:

a) Compute an upper bound $BW^+_{k,\pi}$ on the length of any level-k busy window represented by π;

b) Derive from π_i (the task-level path of G_i in π) and $BW^+_{k,\pi}$ an upper bound $q^+_{v,\pi_i,BW^+_{k,\pi}}$ on the number of jobs of type v released in any level-k busy window represented by π;

c) Compute, for each $q \leq q^+_{v,\pi_i,BW^+_{k,\pi}}$, an upper bound $Q^+_{v,\pi}(q)$ on the length of the q-th queueing prefix of jobs of type v in any level-k busy window represented by π;

d) For each $q \leq q^+_{v,\pi_i,BW^+_{k,\pi}}$, compute a lower bound $\theta^-_{v,\pi_i}(q)$ on the (possibly negative) time difference between the q-th arrival of a job of type v in in any level-k busy window represented by π and the start of that busy window;

e) Based on the above, compute an upper bound $R^+_\pi(v)$ on the worst-case response time of any job of type v in any level-k busy window represented by π.

Step 3: Finally, find an upper bound $R^+_\Sigma(v)$ on $wcrt(v)$ by computing the maximum value using all system paths.

In the following subsection, we provide the formulas used for the computations associated with each step of the methodology that we just presented, along with

some intuition of where they come from. The proof of correctness of the computed values is presented in Section 5.6.

5.5.2 Step-by-step RTA of Gd systems

Our RTA relies on an upper bound on the workload of a set of jobs in any level-p busy window by the *workload for the path* in $\mathbf{\Pi}^v_\Sigma$ that represents it, where the workload of a given set of vertices $V \subseteq V_\Sigma$ for a system-level path $\pi := \{\pi_1, \ldots, \pi_n\} \in \mathbf{\Pi}^v_\Sigma$ and a duration Δ is

$$wl^+_{V,\pi}(\Delta) := \sum_{x=1}^{n} wl^+_{V,\pi_x}(\Delta) \tag{5.3}$$

with $$wl^+_{V,\pi_x}(\Delta) := cost(|pre_{\Delta+\mathbf{J}(fst(\pi_x))}(\pi_x)|_V)$$

We will show (see Lemma 15 in Section 5.6) that the workload of a set of vertices $V \subseteq V_\Sigma$ for any prefix of length Δ of a level-k busy window represented by a system path π is upper bounded by $wl^+_{V,\pi}(\Delta)$.

Step 1: Computing $\mathbf{\Pi}^v_\Sigma$ — *see Theorem 7 in Section 5.6*

Let $\Pi^v_x(\Delta)$ denote a set of paths of task $G_x \in \Sigma$ in which each path π_x is the longest path fitting in $\Delta + \mathbf{J}^+_x$, that is, such that $len(\pi_x) \leq \Delta + \mathbf{J}^+_x$ and for all valid suffixes s, $len(\pi_x \cdot s) > \Delta + \mathbf{J}^+_x$; where $\mathbf{J}^+_x$ representing the largest release jitter among V_x. Note that if the vertex under study v belongs to G_i then we consider only paths π_x with occurrences of v.

With this notion of path, we compute a sufficiently large duration which can bound the length of any level-k busy window. That duration is the least positive fixed point, written $\mathbf{W}_k$, of the following equation

$$\Delta = N + \sum_{x=1}^{n} \max_{\pi_x \in \Pi^v_x(\Delta)} \{wl^+_{hep(k),\pi_x}(\Delta)\} \tag{5.4}$$

with N denoting the greatest non-preemptable segment in the system. At each iteration, the greatest workload among all possible paths within Δ is selected. The obtained fixed point is clearly an upper bound on the duration of any possible busy window. Therefore, to bound $wcrt(v)$, it is sufficient to examine all the following combinations,

$$\mathbf{\Pi}^v_\Sigma := \bigtimes_{x=1}^{n} \Pi^v_x(\mathbf{W}_k) \tag{5.5}$$

where $\bigtimes_{x=1}^{n}$ denotes the Cartesian product of all paths of length bounded by $\mathbf{W}_k$ for the n tasks. Thus, any level-p busy window can be represented by (possibly a prefix of) a path in $\mathbf{\Pi}^v_\Sigma$.

Step 2: Computing upper bounds for each path in $\mathbf{\Pi}^v_\Sigma$

We now show how to compute $BW^+_{k,\pi}$, $q^+_{v,\pi_i,BW^+_{k,\pi}}$, $Q^+_{v,\pi}(q)$ and $\theta^-_{v,\pi_i}(q)$ for a given system path $\pi := \{\pi_1, \ldots, \pi_n\} \in \mathbf{\Pi}^v_\Sigma$.

a) *Computing $BW^+_{k,\pi}$ — see Theorem 8 in Section 5.6*

Having non-preemptable segments implies that vertices in $lp(k)$ may execute within a level-k busy window. Still, the definition of a level-k busy window implies that:

- at most one non-preemptable segment of a vertex in $lp(k)$ can execute in a level-k busy window; and
- such a segment (if it exists) must have started its execution before the beginning of the level-k busy window.

As a result, the maximum duration that vertices in $lp(k)$ can execute in a level-k busy window is upper bounded by:

$$b_k(t_1, k) := \max_{\substack{v_x \in lp(k) \\ C \in \vec{\mathbf{C}}(v_x)}} (C - 1) \tag{5.6}$$

with the convention that $b_k(t_1, k) := 0$ if $lp(k) = \emptyset$. Now, let $BW^+_{k,\pi}$ be the least positive fixed point of the following equation:

$$\Delta = b_k(t_1, k) + wl^+_{hep(k),\pi}(\Delta) \tag{5.7}$$

Then $BW^+_{k,\pi}$ is an upper bound on the length of any possible level-k busy window represented by π.

Note that it may be pessimistic to use $b_k(t_1, k)$ to bound the workload from $lp(k)$ because the largest non-preemptable segment among $lp(k)$ may not contribute to any level-k busy window represented by π. On the other hand, it reduces the complexity of the analysis and simplifies its correctness proof [4].

b) *Computing $q^+_{v,\pi_i,BW^+_{k,\pi}}$*

Definition 40 implies that the number of jobs of type v in a busy window is equal to the number of v in its representing path. In addition, since $BW^+_{k,\pi}$ is an upper bound on the length of any possible level-k busy window represented by π, let $q^+_{v,\pi_i,BW^+_{k,\pi}}$ be the number of vertices v in the prefix $pre_{BW^+_{k,\pi}+\mathbf{J}(fst(\pi_i))}(\pi_i)$ of path π_i, then $q^+_{v,\pi_i,BW^+_{k,\pi}}$ is an upper bound on the number of jobs of type v released in any level-k busy window represented by π.

[4] An exact analysis is easy to define but it would require considering up to $n+1$ different critical instants for a system with n tasks.

c) *Computing* $Q^+_{v,\pi}(q)$ — *see Theorem 9 in Section 5.6*

The q-th queueing prefix of jobs of type v in a level-k busy window represented by π spans the execution of:

- at most one non-preemptable segment from a vertex in $lp(k)$ which started before that busy window;
- all jobs of task G_i with the same priority as v released during the prefix, up to the q-th job of v in π_i (minus the cost of its last segment);
- the jobs of vertices from G_i in $hp(k)$ released during the prefix; and
- the jobs of vertices in $hep(k)$ released during the prefix, except those from G_i.

Let $Q^+_{v,\pi}(q)$ be the least positive fixed point of the following equation:

$$\begin{aligned}\Delta = b_k(t_1, k) \\ &+ wl^+_{ep(k)\cap V_i,\pi}(len^q_v(\pi_i) + 1) - last(\vec{\mathbf{C}}(v)) + 1 \\ &+ wl^+_{hp(k)\cap V_i,\pi}(\Delta) \\ &+ wl^+_{hep(k)\setminus V_i,\pi}(\Delta)\end{aligned} \tag{5.8}$$

where $last(\vec{\mathbf{C}}(v))$ is the maximum cost of the last segment of v. Then, $Q^+_{v,\pi}(q)$ is an upper bound on the length of the q-th queueing prefix of jobs of type v in any possible level-k busy window represented by π.

d) *Computing* $\theta^-_{v,\pi_i}(q)$ — *see Theorem 10 in Section 5.6*

For each $q \leq q^+_{v,\pi_i,BW^+_{k,\pi}}$, a lower bound on the duration between t_1 and the q-th arrival of jobs of type v in any possible level-k busy window represented by π is:

$$\theta^-_{v,\pi_i}(q) := len^q_v(\pi_i) - \mathbf{J}(fst(\pi_i)) \tag{5.9}$$

e) *Computing* $R^+_\pi(v)$ — *see Theorem 11 in Section 5.6*

The worst-case response time $R^+_\pi(v)$ for path π can be upper bounded based on the above upper and lower bounds.

$$R^+_\pi(v) := \max_{q \leq q^+_{v,\pi_i,BW^+_{k,\pi}}} \{Q^+_{v,\pi}(q) - \theta^-_{v,\pi_i}(q) + last(\vec{\mathbf{C}}(v)) - 1\} \tag{5.10}$$

Step 3: Computing $R^+_\Sigma(v)$

Finally, performing the same computation for all paths in $\mathbf{\Pi}^v_\Sigma$ yields the worst-case response time of jobs of type v.

$$R^+_\Sigma(v) := \max_{\pi \in \mathbf{\Pi}^v_\Sigma} (R^+_\pi(v)) \tag{5.11}$$

Note that the max functions in Equation 5.10 and 5.11 play different roles. The first one (Equation 5.10) accounts for the fact that there may be several jobs of the same type (vertex) in a level-k busy window. The second one (Equation 5.11) allows a fine-grained analysis of dependencies which are captured by the notion of path.

5.5.3 Improvements

For the sake of clarify, we have presented the analysis by first computing a superset of possible paths and then focusing on a single vertex. Clearly, this approach is not the most efficient. First, the paths considered are larger than needed and the analysis is likely to consider many times the same prefix common to many paths. Second, as presented, a complete system analysis would need to iterate the same process for each vertex. Both points entail costly and/or useless recomputations.

A more reasonable approach is to analyse pertinent paths only once. An analysis of the whole system would proceed by considering all possible alignments between vertices of all tasks. For each alignment $\{v_1, \ldots, v_n\}$, we compute the level-p busy window (with p the minimal priority) for all possible paths starting from the considered alignment. Paths are built on demand; they end when a level-p quiet time is reached. That busy window is the longest and includes all other busy windows. The computation should keep enough information to evaluate $Q^+_{v,\pi}(q)$, $\theta^-_{v,\pi_i}(q)$ and therefore $R^+_\pi(v)$ for each vertex v in the path. Finally, the maximum of $R^+_\pi(v)$ over all alignments gives the worst-case response time for any vertex v (*i.e.,* jobs of type v).

Further improvements can be made. Consider a vertex v_i in an alignment $\{v_1, \ldots, v_n\}$, then if there is some v_j with a lower priority than v_i whereas task G_j has another vertex with a priority equal or higher than v_i, then this alignment cannot be a worst case for v_i and $R^+_\pi(v_i)$ does not need to be computed in that alignment.

The analysis described so far is precise; the only source of approximation lies in the blocking factor $b_k(t_1, k)$ (see Equation 5.6). However, depending on the size of GD, considering all possible alignments may be overly expensive. Approximations should be studied. A possible approximation consists in computing a single over-approximated workload function for each priority and each task. This would lower drastically the number of combinations to test. Note that this approach is similar to the approximation used by Tindell in his offset analysis [143] and by Guan *et al.* in their DRT analysis [69]. The optimization techniques presented in Section 4.5 can be applied to this RTA without degrading the precision of the result.

5.6 Proof of Correctness

We outline here the proofs of the main lemmas used to establish the correctness of the RTA presented in Section 5.5. Future works would be to formalize these proofs using Coq and the Prosa library. We present here these proofs in a way that is amenable to their formalizations in Coq. In contrast with classical proofs in the RT community, machine-verified proofs require to list all used hypotheses, to specify formally concrete executions and to prove many properties usually taken for granted.

In the following, we consider a JFPLP schedule σ and a system-level job arrival sequence $\rho \sim \Sigma$ having a job $\jmath$ of type v with priority k and belonging to task G_i. Any such job occurs within a level-k busy window. Therefore, we consider that $\jmath$ occurs as the q-th job of type v in a level-k busy window starting at instant t_1.

5.6.1 Concrete busy window and queueing prefix

The key to the proof of the RTA is to show the correctness of the upper/lower bounds (*e.g.,* $BW^{+}_{p,\pi}$) computed using the abstract model (*i.e.,* GD). So, we need to specify precisely the length BW_k of the considered level-k busy window and the length $Q_{v,\rho}(q)$ of the q-th queueing prefix of jobs of type v in order to bound them.

Length of the level-k busy window

The length of a level-k busy window starting at t_1, denoted by BW_k, can be computed as the least positive fixed point of the following equation.

$$\Delta = serv_{lp(k),\sigma}(t_1, \Delta) + wl_{hep(k),\rho}(t_1, \Delta) \tag{5.12}$$

The first term represents the service provided to the possible non preemptable segment of a lower priority job. Then, the amount of services performed by the scheduler for $hep(k)k$ jobs is equal to the workload requested from $hep(k)k$ within the duration of the busy window.

Length of the q-th queueing prefix

Similarly, computing the length of the q-th queueing prefix of jobs of type v (*i.e.,* the queueing prefix of $\jmath$) in the level-k busy window $[t_1, t_1 + BW_k[$ amounts to evaluate:

- the service provided to the possible non preemptable segment of a lower priority job;

- the workload of jobs of task G_i with the priority k (minus the cost of j's last segment);
- the workload of jobs of vertices from G_i in $hp(k)k$;
- the workload of other jobs with priorities in $hep(k)k$.

Thus, the length of the q-th queueing prefix of jobs of type v, denoted by $Q_{v,\rho}(q)$, can be defined as the least positive fixed point of the following equation.

$$\begin{aligned}\Delta = & serv_{lp(k),\sigma}(t_1, \Delta) \\ & + wl_{ep(k)\cap V_i,\rho}(t_1, (\mathbf{a}(j) - t_1 + 1)) - last(\vec{\mathbf{c}}(j)) + 1 \\ & + wl_{hp(k)\cap V_i,\rho}(t_1, \Delta) \\ & + wl_{hep(k)\setminus V_i,\rho}(t_1, \Delta)\end{aligned} \tag{5.13}$$

5.6.2 Basic lemmas

To bound BW_k and $Q_{v,\rho}(q)$ we rely on the following lemma.

Using Lemma 3 proved in Chapter 4, the proof that $BW^+_{k,\pi}$ and $Q^+_{v,\pi}(q)$ upper bound BW_k and $Q_{v,\rho}(q)$ respectively, amounts to show that $serv$ and wl are increasing and to compare the rhs of their recursive definitions.

A key step to this end is to bound the workload of a set of jobs of types in V during a busy window by the workload of the set of vertices in V of the path representing that busy window. More generally, we prove the following lemma:

Lemma 15. *Let Δ be a time interval and let $\pi := \{\pi_1, \ldots, \pi_n\}$ be the system path representing $\hat{\rho}_{/[t_1,t_1+\Delta[}$* i.e., *$\hat{\rho}_{/[t_1,t_1+\Delta[} \sim \pi$, then*

$$wl_{V,\rho}(t_1, \Delta) \leq wl^+_{V,\pi}(\Delta) \tag{5.14}$$

Furthermore, that bound is tight.

Proof. The arrival sequence ρ can be decomposed into n independent task-level job arrival sequences $\{\rho_1, \ldots, \rho_n\}$ where each ρ_x is the arrival sequence of the jobs of task G_x. Then,

$$wl_{V,\rho}(t_1, \Delta) = \sum_{x=1}^{n} wl_{V,\rho_x}(t_1, \Delta)$$

Recall the definition of

$$wl^+_{V,\pi}(\Delta) := \sum_{x=1}^{n} cost(|pre_{\Delta+\mathbf{J}(fst(\pi_x))}(\pi_x)|_V)$$

Therefore, it is sufficient to prove, for all tasks G_x that

$$wl_{V,\rho_x}(t_1, \Delta) \leq cost(|pre_{\Delta+\mathbf{J}(fst(\pi_x))}(\pi_x)|_V)$$

We show that the workload of $\hat{\rho}_{x/[t_1,t_1+\Delta[}$ is maximal when:

1. any two consecutive jobs of ρ_x are separated by their minimum inter-arrival time d;
2. all jobs take their maximum cost (*i.e.,* WCET) $\mathbf{C}$;
3. the first job in the interval releases at t_1 after having experienced its maximum release jitter ($\mathbf{J}(fst(\pi_x))$) whereas the jitter of all other jobs is null.

When these three conditions are met, $wl_{V,\rho_x}(t_1, \Delta)$ is maximal and exactly equal to $cost(|pre_{\Delta+\mathbf{J}(fst(\pi_x))}(\pi_x)|_V)$. That directly implies Lemma 15. □

5.6.3 Correctness of system paths

As a first step towards the correctness proof of the RTA, we must show that any possible level-p busy window is represented by one path in $\mathbf{\Pi}^v_\Sigma$.

Theorem 7. *Any level-k busy window is represented by at least one path in $\mathbf{\Pi}^v_\Sigma$ as defined in Equation 5.5.*

Proof. It suffices to show that $\mathbf{W}_k$ bounds the length of any level-k busy window. *i.e.,*

$$BW_k \leq \mathbf{W}_k \tag{5.15}$$

Recall that $\mathbf{W}_k$ is the least positive fixed point of the equation

$$\Delta = N + \sum_{x=1}^{n} \max_{\pi_x \in \Pi^v_x(\Delta)} \{wl^+_{hep(k),\pi_x}(\Delta)\} \tag{5.16}$$

We first prove that the *rhs* of Equation 5.16 bounds the one of Equation 5.17. It is fairly easy to prove that $b_k(t_1, k) \leq N$ since N represents the largest segment in the system. Furthermore, for each task $G_x \in \Sigma$, we clearly have

$$wl^+_{hep(k),\pi_x}(\Delta) \leq \max_{\pi_x \in \Pi^v_x(\Delta)} \{wl^+_{hep(k),\pi_x}(\Delta)\}$$

and therefore, for the system path and all tasks,

$$wl^+_{hep(k),\pi}(\Delta) \leq \sum_{x=1}^{n} \max_{\pi_x \in \Pi^v_x(\Delta)} \{wl^+_{hep(k),\pi_x}(\Delta)\}$$

Then, Lemma 3 entails $BW^+_{p,\pi} \leq \mathbf{W}_k$ and Theorem 8 permits to establish Equation 5.5 by transitivity. □

5.6.4 Correctness of bounds for jobs

Now, we show the correctness of the upper/lower bounds $BW^+_{p,\pi}$, $q^+_{v,\pi_i,BW^+_{k,\pi}}$, $Q^+_{v,\pi}(q)$ and $\theta^-_{v,\pi_i}(q)$ which implies the correctness of the upper-bound $R^+_\pi(v)$ for the response time of $\jmath$. Let π be the path representing the level-k busy window $[t_1, t_1 + BW_k[$,

Theorem 8. *Let $BW^+_{p,\pi}$ be the least positive fixed point of the equation*

$$\Delta = b_k(t_1, k) + wl^+_{hep(k),\pi}(\Delta) \tag{5.17}$$

$$\textit{then} \qquad BW_k \leq BW^+_{p,\pi} \tag{5.18}$$

Proof. We first prove that the *rhs* of Equation 5.17 bounds the *rhs* of

$$\Delta = serv_{lp(k),\sigma}(t_1, BW_k) + wl_{hep(k),\rho}(t_1, \Delta) \tag{5.19}$$

The definition of a level-p busy window implies that at most one non-preemptable segment of any vertex in $lp(k)$ of π can execute in $[t_1, t_1 + BW_k[$. Further, such a segment (if it exists) must have started execution before t_1. Therefore, $serv_{lp(k),\sigma}(t_1, BW_k)$ is bounded by $b_k(t_1, k)$. Lemma 15 ensures that the second term of the *rhs* of Equation 5.19 is bounded by $wl^+_{hep(k),\pi}(\Delta)$. Then, Lemma 3 permits to conclude. □

Lemma 16. *The number of vertices v in π_i upper-bounds q:*

$$q \leq q^+_{v,\pi_i,BW^+_{k,\pi}} \tag{5.20}$$

Proof. This result follows by the definition of a path representing a busy window and $q^+_{v,\pi_i,BW^+_{k,\pi}}$. □

Theorem 9. *Let $Q^+_{v,\pi}(q)$ be the least positive fixed point of the equation*

$$\begin{aligned} \Delta = {} & b_k(t_1, k) \\ & + wl^+_{ep(k)\cap V_i,\pi}(len^q_v(\pi_i) + 1) - last(\vec{\mathbf{C}}(v)) + 1 \\ & + wl^+_{hp(k)\cap V_i,\pi}(\Delta) \\ & + wl^+_{hep(k)\setminus V_i,\pi}(\Delta) \end{aligned} \tag{5.21}$$

then it bounds the length of the q-th queueing prefix of jobs of type v in the level-k busy window i.e.,

$$Q_{v,\rho}(q) \leq Q^{+}_{v,\pi}(q) \tag{5.22}$$

Proof. As for Theorem 8, it suffices to prove that the *rhs* of Equation 5.21 bounds the *rhs* of Equation 5.13. We prove it by considering each term in turn.

1. In Theorem 8, we proved $serv_{lp(k),\sigma}(t_1, BW_k) \leq b_k(t_1, k)$. Further, the definition of a queueing prefix implies that $Q_{v,\rho}(q) \leq BW_k$. Therefore, for all $\Delta \leq Q_{v,\rho}(q)$ (which is sufficient for Lemma 3), $serv_{lp(k),\sigma}(t_1, \Delta) \leq serv_{lp(k),\sigma}(t_1, BW_k) \leq b_k(t_1, k)$.
2. The second term can be divided in two parts:

$$wl^{+}_{ep(k)\cap V_i,\pi\setminus v^q}(len^q_v(\pi_i) + 1) \tag{5.23}$$
$$+ \mathbf{C}(v) - last(\vec{\mathbf{C}}(v)) + 1 \tag{5.24}$$

   where v^q denotes the q-th v in π.

   The second term of Equation 5.13 can be separated in two parts as well:

$$wl_{ep(k)\cap V_i,\rho\setminus\jmath}(t_1, \mathbf{a}(\jmath) - t_1 + 1) \tag{5.25}$$
$$+ \mathbf{c}(\jmath) - last(\vec{\mathbf{c}}(\jmath)) + 1 \tag{5.26}$$

   Now, it is sufficient to prove that (5.25) $\leq$ (5.23) and (5.26) $\leq$ (5.24).

   - According to Definition 40, the type of any job of $hep(k)k \cap V_i$ released in $[t_1, \mathbf{a}(\jmath) + 1[$ appears in $pre_{\mathbf{J}(fst(\pi_i))+len^q_v(\pi_i)+1}(\pi_i)$. Taking into account the minimum inter-arrival time, we have

$$wl_{ep(k)\cap V_i,\rho}(t_1, \mathbf{a}(\jmath) - t_1 + 1) \leq wl^{+}_{ep(k)\cap V_i,\pi}(len^q_v(\pi_i) + 1)$$

     If we filter out $\jmath$ from ρ and the q-th v (job $\jmath$'s type) from π, (5.25) $\leq$ (5.23) follows.
   - By definition, we know that $\vec{\mathbf{c}}(\jmath) = \langle c_1, \ldots, c_s \rangle$, $\vec{\mathbf{C}}(v) = \langle C_1, \ldots, C_s \rangle$ and, for all $i = 1, \ldots, s$, $c_i \leq C_i$. So

$$\sum_{i=1}^{s-1} c_i \leq \sum_{i=1}^{s-1} C_i$$

     Equivalently $(\mathbf{c}(\jmath) - last(\vec{\mathbf{c}}(\jmath)) \leq (\mathbf{C}(v) - last(\vec{\mathbf{C}}(v))$. Therefore (5.26) $\leq$ (5.24) follows.

The last two inequalities between the last two terms of (5.21) and of (5.13) follow directly from Lemma 15. □

Theorem 10. *The term* $\theta^{-}_{v,\pi_i}(q) = len^{q}_{v}(\pi_i) - \mathbf{J}(fst(\pi_i))$ *is a lower bound of the duration between the arrival of* $\jmath$ *and the beginning of its level-k busy window.*

$$\theta^{-}_{v,\pi_i}(q) \leq \mathbf{a}(\jmath) - t_1 \tag{5.27}$$

Proof. By cases.

(1) $\mathbf{a}(\jmath) - \mathbf{t_1} < \mathbf{0}$. The job $\jmath$ arrives before t_1 but releases at or after t_1 *i.e.,* $t_1 \leq \mathbf{a}(\jmath) + \mathbf{j}(\jmath)$. Constrained jitter implies that no other job of the same task arrives before the release of $\jmath$. So job $\jmath$ must be the first vertex of π_i and $len^{q}_{v}(\pi_i) = 0$. By convention, $\mathbf{j}(\jmath) \leq \mathbf{J}(v)$, therefore $-\mathbf{J}(v) \leq -\mathbf{j}(\jmath) \leq \mathbf{a}(\jmath) - t_1$ and Equation 5.27 holds.

(2) $\mathbf{a}(\jmath) - \mathbf{t_1} \geq \mathbf{0}$. Let $\jmath'$ be the job corresponding to the first vertex in π_i. We know $\mathbf{a}(\jmath) - \mathbf{a}(\jmath') \geq len^{q}_{v}(\pi_i)$ and $\mathbf{j}(\jmath') \leq \mathbf{J}(fst(\pi_i))$. Then, $len^{q}_{v}(\pi_i) - \mathbf{J}(fst(\pi_i)) \leq \mathbf{a}(\jmath) - \mathbf{a}(\jmath') - \mathbf{j}(\jmath')$ and since $\mathbf{a}(\jmath') + \mathbf{j}(\jmath') \geq t_1$ that is $-\mathbf{a}(\jmath') - \mathbf{j}(\jmath') \leq t_1$, Equation 5.27 follows. □

Lemma 17. *Let* $R_\jmath$ *be the response time of job* $\jmath$ *then*

$$R_\jmath \leq Q^{+}_{v,\pi}(q) - \theta^{-}_{v,\pi_i}(q) + last(\vec{\mathbf{C}}(v)) - 1 \tag{5.28}$$

Proof. By definition $R_\jmath := \mathbf{end}(\jmath) - \mathbf{a}(\jmath)$. Also, when a job begins to execute its last non-preemtable segment it cannot be preempted until its completion. Using the notion of q-th queueing prefix, we have

$$\mathbf{end}(\jmath) = t_1 + Q_{v,\rho}(q) + last(\vec{\mathbf{c}}(\jmath)) - 1$$

Note that $Q_{v,\rho}(q)$ includes the first cost unit of job $\jmath$'s last segment, so we should subtract 1 from $last(\vec{\mathbf{c}}(\jmath))$. Equivalently, we have

$$R_\jmath = Q_{v,\rho}(q) - (\mathbf{a}(\jmath) - t_1) + last(\vec{\mathbf{c}}(\jmath)) - 1$$

The result follows from Theorem 9, Theorem 10 and the fact that costs in the model upper-bounds costs in the execution. □

5.6.5 Correctness of the RTA

The previous result applies to an arbitrary job $\jmath$ of type v in a busy window starting at t_1. It can be used to upper-bound the response times of any job of type v in that busy window, and by extension any job of type v in the arrival sequence.

Theorem 11. *The response time of any job of type v released in a busy window starting at t_1 is bounded by*

$$\max_{q \leq q^+_{v,\pi_i, BW^+_{k,\pi}}} \{Q^+_{v,\pi}(q) - \theta^-_{v,\pi_i}(q) + last(\vec{\mathbf{C}}(v)) - 1\} \tag{5.29}$$

Proof. Follows from properties of max and Lemma 17. □

Theorem 12. *The response time of any job of type v released in any arrival sequence $\rho \sim \Sigma$ is bounded by*

$$\max_{\pi \in \mathbf{\Pi}^v_\Sigma} \left\{ \max_{q \leq q^+_{v,\pi_i, BW^+_{k,\pi}}} \left\{ Q^+_{v,\pi}(q) - \theta^-_{v,\pi_i}(q) + last(\vec{\mathbf{C}}(v)) - 1 \right\} \right\} \tag{5.30}$$

Proof. Follows from properties of max and Theorem 11. □

5.7 Towards Formal Verification

In this section, we discuss the significance of the presented model and analysis in the context of our broader effort toward a Coq library of schedulability results.

As presented early in Chapter 2, the Prosa library [30, 29] has been proposed to provide formal specifications and mechanized proofs for schedulability analyses using the Coq proof assistant. The motivation behind our general task model for fixed priority scheduling is to add it to the Prosa library and prove the correctness of its RTA. It can thus cover a large variety of existing models and analyses.

5.7.1 Proving in Coq the RTA of GD systems

The complete Coq proof of the RTA for GD systems is still in progress and can be separated into two parts:

1. The generic proof of RTAs for the JFPLP scheduling policy. For this part, many definitions (*i.e.,* those in Section 5.2) have been formalized and used in Prosa, as well as a significant part of the proof. We still need to formalize the proof of a

more general statement, which does not rely on a task model, but on an abstract workload function

$$wl^+ : (\rho \to \mathbb{N} \to \mathbb{N} \to T) \to \mathbb{N} \to \mathbb{N}$$

where: (a) the first argument $(\rho \to \mathbb{N} \to \mathbb{N} \to T)$ denotes a function taking a job arrival sequence, a time instant and a time duration and returning an abstract *candidate* represented by the type T, where candidates correspond to incomparable scenarios which must be analyzed, *i.e.,* paths for the GD model; (b) the second argument denotes a time duration such that wl^+ returns the workload during that duration. The proof as well as the analysis are then applicable to many task models respecting the fixed priority scheduling policy, including the GD model, by instantiating that function.

2. Specifying the GD task model and instantiating the function wl^+. The key of instantiating the function wl^+ is to find a system path which maximizes the actual workload requested by a given arrival sequence within a given duration. Consider a system Σ and a set of its system paths Π_Σ, to validate this instantiation, we have to prove that for any arrival sequence $\rho \sim \Sigma$ and a given duration Δ, there exists a system path $\pi \in \Pi_\Sigma$ such that $wl_{V,\rho}(0, \Delta) \leq wl^+_{V,\pi}(\Delta)$. Which ensures that that set of system paths can cover any cases of system behavior and can be used for computing an upper bound of the response time of any vertex in the system using the RTA algorithm presented in Section 5.5.

5.7.2 Intended use of the analysis

One of our objectives is to formally certify our RTA in order to:

- obtain certified existing RTAs for task models which can be expressed by GD. As GD is very expressive, we can instantiate all more precise task models into it then obtain their certified RTAs by proving the correctness of their instantiations *i.e.,* a more specif model and its instantiation into GD share the same workload functions (scenarios);
- obtain certified RTAs for task models beyond GD by reusing the generic part of the proof. For those task models that cannot be expressed by GD, we can formally specify them and instantiate them into the abstract workload functions, then to get certified RTAs. This can reduce many proof efforts;
- obtain certified approximate RTAs by reusing the generic proofs part and focusing on upper bounding the workload. For a task model, we can get different approximation-level RTAs by proposing different sets of workload

functions, which can be used to instantiate the abstract workload functions. A set of workload functions is validated if these workload functions can cover or dominate all possible scenarios;

- obtain many certified result certifiers like CertiCAN using certified RTAs. To this goal, one main concern is the efficiency of certified tools, and we may need more proof efforts to optimize them and to make them scalable.

5.7.3 Beyond the current analysis

By proposing a unified analysis for models as different as the DRT model and Tindell's offset model, our work underlines the generic parts of the proof structure of such RTAs. Based on this, we can now propose a framework which formalizes these steps in a generic manner, to be reused for any new task model. Such steps include the use of sustainability properties [12, 30], but also strategies to efficiently approximate the worst-case response time.

Note that the schedulability analysis of the GD task model for the JLFPNP policy is a *strongly coNP-hard* problem because this model is a generalized version of the DRT task model whose FP schedulability analysis has been proven coNP-hard in the strong sense [128].

5.8 Related Work

Many task models have been proposed to analyze different fixed-priority scheduling policies. Depending on their capacity to model intra- or inter-task dependencies, those models can be divided into three categories.

The simplest models do not consider any kind of dependency: there is only one type of job for each task. The classic *periodic* task model, presented by Liu and Layland [96] characterizes a task by its worst-case execution time C and its activation period P. The *sporadic* task model [106] generalizes activation periods of tasks by introducing the concept of minimum inter-arrival time. Later Thiele et *al.* introduced the *real-time calculus* [141], whose *arrival curves* can model many more arrival patterns.

Another category of models considers *intra-task* dependencies: there may be several types of jobs for each task. The *multiframe* model [105], characterizes a task by an array of execution times $(C^0, C^1, \ldots, C^{N-1})$ and a minimum inter-arrival time P. A task has N types of jobs and the $(i+1)$-st job in the arrival sequence has the worst-case execution time $C^{(i \bmod N)}$ and arrives at least P time units af-

ter the arrival time of the i-th job. The model $(\boldsymbol{C}, \boldsymbol{P}, \boldsymbol{D})$ extends the multiframe model by allowing each type of job to have a different inter-arrival time and deadline [14]. Later, Baruah introduced the *recurring branching* task model [15], which adds branching structures. Each task can be represented by a tree of job types $(\boldsymbol{C}, \boldsymbol{D})$ labeled by minimum inter-arrival times. Two other extensions use *directed acyclic graphs* instead of trees: the *recurring real-time* task model [16, 17]), and the *non-cyclic recurring real-time* task model [13]. More recently, Stigge et *al.* introduced the DRT task model [129] and its extended version [130] that use arbitrary graphs. None of those models allows to model inter-task dependencies.

One of the most classic task model taking *inter-task* dependencies into account is Tindell's *offsets* model. A system is made of a collection of transactions regrouping periodic tasks having fixed timing relations. More recently, the DRT model was extended by Mohaqeqi et al. to allow the RDV mechanism [104] and by Abdullah et al. to take into account shared resources [1].

To the best of our knowledge, none of the previous models is general enough to express at the same time intra- and inter-task dependencies as well as arrival curves. Our model is a generalization in this respect and is able to express all the above task models. The formal certification of its associated RTA should permit to factorize the correctness proofs of many analyses.

5.9 Conclusion

In this chapter, we have introduced the GD model, a generalization of the DRT task model that is expressive enough to model and analyze many different fixed-priority systems. In particular, GD can express dependencies between jobs as well as tasks. The work presented in this chapter is motivated by our ongoing contribution to Prosa, a Coq library of models and analyses of real-time systems. The GD model and its associated RTA provide the needed foundations for a Coq certified response time analysis of complex systems, in particular regarding dependencies.

Chapter 6

Conclusion

6.1 Summary

In this thesis, we have used theorem proving to provide high confidence in hard real-time systems schedulability analysis as well as related industrial tools. In particular, we have considered and addressed three open problems:

- **Consistency of formal abstract models.** Theorem proving can formally guarantee the correctness of analyses because it permits to mechanically check all corresponding proofs. However, the consistency of the specifications of the model needs to be manually checked. Using theorem proving, system models are often specified in an abstract and analysis-convenient manner. The adequacy of these models to reality remains to be justified. It is not always clear that the analyses proven over a high-level model can be applied to a lower-level or concrete model.

  To address this problem, Chapter 3 proposed a formal interface using the Coq proof assistant. It combines the schedulability analyses proven in the Prosa library with a formally verified concrete OS kernel, RT-CertiKOS. In Prosa, the abstract system model is suitable for reasoning about schedulability analysis, but its consistency is not established using a real system. This work used a concrete schedule from RT-CertiKOS to validate Prosa's abstract model. It showed that analyses proven over an abstract and analysis-convenient model can be applied to a concrete system. This work also provided RT-CertiKOS with a modular, state-of-the-art schedulability proof. Our interface is general enough to be used for integrating other analyses from Prosa into RT-CertiKOS. We believe that this work provides new insights into the formal connections between two independent mechanized proof projects.

- **Certification of industrial tools.** Schedulability analysis aims at guaranteeing the absence of deadline misses for hard real-time systems. This property is crucial for the systems used in safety-critical domains such as avionics and automotive. To provide these guarantees, many schedulability analyses have been implemented in industrial tools. However, there is no formal guarantees that these tools are correct because the underlying analyses may be flawed or their implementation may contain undetected bugs.

  To address this problem, Chapter 4 proposed to use result certification, instead of tool certification. Specifically, using Coq, we formally verified a result certifier, named CertiCAN, for certifying the results produced by industrial CAN analyzers. Result certification is a flexible and light-weight process which is suitable for industry practice since it does not need the source code and is not impacted by software updates. The idea is to: 1) combine a precise and approximate analyses for time efficiency; 2) prove them correct using Coq; 3) extract an Ocaml result certifier from proofs using Coq's extraction technique. Our experiment shows that CertiCAN is efficient enough and can be used to certify the results produced by the industrial tool RTaW-Pegase even for large systems.

- **Factorization of formal proofs.** Formal verification of RTAs requires a significant proof effort. Typically, the formal verification of a single analysis takes several person weeks. An obvious question is then how to factorize the proofs to make them generic and general enough so that they can be reused by several analyses?

  To address this problem, Chapter 5 proposed GD, a very general task model and its corresponding RTA amenable to Coq formalizations. The model is based on the DRT model and extends it with several additional parameters such as jitters, non-preemptable segments. GD is expressive enough to model many kinds of systems. The idea is to formally verify a general RTA for this model. Then, proving the correctness of a specific RTA boils down to proving its instantiation to this model. This should reduce the proof effort.

6.2 Future Work

In this last section, we discuss several further research directions this thesis suggests.

- **General models, analyses, and proofs.** Â straightforward future work is to complete the Coq formalization of the RTA of GD. It would permit

a formal comparison of our proposed analysis in Chapter 5 with the existing RTAs of specific types of models, *e.g.,* constrained deadline with job-level FPP or task-level FPNP [132], and arbitrary deadline with task-level FPP [111]. As the generic part of this RTA permits to instantiate many RTAs with different approximation levels, it would be interesting to conduct a practical study of the complexity of the analysis and of the possible trade-off between accuracy of the computed bounds and runtime performance of an RTA implementation. Then, using the same ideas as presented in Chapter 4, the formally verified RTAs would permit to extract many result certifiers. Whether these extracted tools will be efficient enough for industry practice is an open question.

Another research topic is to study the theoretical connection between the RTA proposed in Chapter 5 and the notion of sustainability. Both RTA and sustainability analysis can be abstracted as a scenario-based analysis. The connection between them is the exact worst-case scenarios. For instance, for a system, the property that an RTA is sustainable with respect to a parameter P is defined as the system that is schedulable according to the RTA being still schedulable when parameter P becomes *better*. In other words, the worst-case scenarios would not be worse when parameter P becomes better. The idea of RTA is to upper bound the exact worst-case scenarios. We believe that a formal connection of these two concepts would allow us to build a fundamental and general structure for schedulability analysis.

- **Regarding result certification.** CertiCAN shows that result certification is possible and suitable for industry practice. It is, however, still time consuming to obtain a verified result certifier using a theorem prover. CertiCAN requires many optimizations to be proved to reach sufficient efficiency. Therefore, an interesting research problem is to find simpler and more efficient certification strategies. CertiCAN only requires the results produced by industrial tools as its input. We should study what kind of additional information industrial tools could provide to reduce the proof effort and increase efficiency of result certifiers. For instance, CertiCAN computes a lot of fixed points of workload functions, and it needs several iterations to find a fixed point. If industrial tools could provide these fixed points, CertiCAN would only need to check their correctness and it would increase efficiency. Another additional information would could be useful is the worst scenarios considered by industrial tools.

Appendix A

A.1 NETCARBENCH Configuration 1

```
<netcarbench-data version="3.2" >
<can-network name="EVALUATION" granularity="5" bandwidth="500" >
<network-load min="0.4" max="0.6" />
<nb-network-interfaces min="7" max="15" />
<fixed-station-loads>
<station id="1" value="0.30" />
</fixed-station-loads>
<frame-periods>
<period value="5" weight="2" margin="1" prio_low_range="1" prio_high_range="200"/>
<period value="10" weight="5" margin="2" prio_low_range="201" prio_high_range="400"/>
<period value="20" weight="5" margin="2" prio_low_range="401" prio_high_range="600"/>
<period value="50" weight="10" margin="4" prio_low_range="601" prio_high_range="800"/
    >
<period value="100" weight="10" margin="4" prio_low_range="801" prio_high_range="1000
    "/>
<period value="200" weight="5" margin="2" prio_low_range="1001" prio_high_range="1200
    "/>
<period value="500" weight="2" margin="1" prio_low_range="1201" prio_high_range="1400
    "/>
<period value="1000" weight="2" margin="1" prio_low_range="1401" prio_high_range="
    1600"/>
</frame-periods>
<frame-payloads>
<payload value="1" weight="1" margin="1"/>
<payload value="2" weight="1" margin="1"/>
<payload value="3" weight="1" margin="1"/>
<payload value="4" weight="2" margin="1"/>
<payload value="5" weight="3" margin="2"/>
<payload value="6" weight="4" margin="2"/>
<payload value="7" weight="5" margin="2"/>
<payload value="8" weight="6" margin="3"/>
</frame-payloads>
<offsets mode="RANDOM"/>
</can-network>
</netcarbench-data>
```

Listing A.1: Configuration file for NETCARBENCH

A.2 NETCARBENCH Configuration 2

```
<netcarbench-data version="3.2" >
<can-network name="EVALUATION" granularity="5" bandwidth="500" >
<network-load min="0.6" max="0.8" />
<nb-network-interfaces min="15" max="20" />
<fixed-station-loads>
<station id="1" value="0.30" />
</fixed-station-loads>
<frame-periods>
<period value="5" weight="2" margin="1" prio_low_range="1" prio_high_range="200"/>
<period value="10" weight="5" margin="2" prio_low_range="201" prio_high_range="400"/>
<period value="20" weight="5" margin="2" prio_low_range="401" prio_high_range="600"/>
<period value="50" weight="10" margin="4" prio_low_range="601" prio_high_range="800"/
    >
<period value="100" weight="10" margin="4" prio_low_range="801" prio_high_range="1000
    "/>
<period value="200" weight="5" margin="2" prio_low_range="1001" prio_high_range="1200
    "/>
<period value="500" weight="2" margin="1" prio_low_range="1201" prio_high_range="1400
    "/>
<period value="1000" weight="2" margin="1" prio_low_range="1401" prio_high_range="
    1600"/>
</frame-periods>
<frame-payloads>
<payload value="1" weight="1" margin="1"/>
<payload value="2" weight="1" margin="1"/>
<payload value="3" weight="1" margin="1"/>
<payload value="4" weight="2" margin="1"/>
<payload value="5" weight="3" margin="2"/>
<payload value="6" weight="4" margin="2"/>
<payload value="7" weight="5" margin="2"/>
<payload value="8" weight="6" margin="3"/>
</frame-payloads>
<offsets mode="RANDOM"/>
</can-network>
</netcarbench-data>
```

Listing A.2: Configuration file 2 for NETCARBENCH

Bibliography

[1] Jakaria Abdullah, Morteza Mohaqeqi, Gaoyang Dai, and Wang Yi. *Schedulability Analysis and Software Synthesis for Graph-Based Task Models with Resource Sharing*. RTAS, 2018. (Cited on pages 13, 99, 106, and 122.)

[2] Rajeev Agrawal, Rene L Cruz, Clayton Okino, and Rajendran Rajan. Performance bounds for flow control protocols. *IEEE/ACM transactions on networking*, 7(3):310–323, 1999. (Cited on page 27.)

[3] Rajeev Agrawal and Rajendran Rajan. *Performance bounds for guaranteed and adaptive services*. IBM TJ Watson Research Center Yorktown Heights, NY, 1996. (Cited on page 27.)

[4] Rajeev Alur, Costas Courcoubetis, Nicolas Halbwachs, Thomas A Henzinger, Pei-Hsin Ho, Xavier Nicollin, Alfredo Olivero, Joseph Sifakis, and Sergio Yovine. The algorithmic analysis of hybrid systems. *Theoretical computer science*, 138(1):3–34, 1995. (Cited on page 29.)

[5] Rajeev Alur and David L Dill. A theory of timed automata. *Theoretical computer science*, 126(2):183–235, 1994. (Cited on page 29.)

[6] Tobias Amnell, Elena Fersman, Leonid Mokrushin, Paul Pettersson, and Wang Yi. Times: a tool for schedulability analysis and code generation of real-time systems. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 60–72. Springer, 2003. (Cited on page 29.)

[7] June Andronick, Corey Lewis, Daniel Matichuk, Carroll Morgan, and Christine Rizkallah. Proof of os scheduling behavior in the presence of interrupt-induced concurrency. In *International Conference on Interactive Theorem Proving*, pages 52–68. Springer, 2016. (Cited on pages 31 and 58.)

[8] June Andronick, Corey Lewis, and Carroll Morgan. Controlled Owicki-Gries concurrency: Reasoning about the preemptible eChronos embedded operat-

ing system. In *Proceedings Workshop on Models for Formal Analysis of Real Systems, MARS*, pages 10–24, 2015. (Cited on page 58.)

[9] Vard Antinyan. *Revealing the Complexity of Automotive Software*, 08 2018. (Cited on page 3.)

[10] Mickaël Armand, Germain Faure, Benjamin Grégoire, Chantal Keller, Laurent Théry, and Benjamin Wener. *Verifying SAT and SMT in Coq for a fully automated decision procedure*, 2011. (Cited on page 41.)

[11] Theodore P Baker and Michele Cirinei. Brute-force determination of multiprocessor schedulability for sets of sporadic hard-deadline tasks. In *International Conference On Principles Of DIstributed Systems*, pages 62–75. Springer, 2007. (Cited on page 29.)

[12] S. Baruah and A. Burns. Sustainable scheduling analysis. In *2006 27th IEEE International Real-Time Systems Symposium (RTSS'06)*, pages 159–168, Dec 2006. (Cited on page 121.)

[13] Sanjoy Baruah. The non-cyclic recurring real-time task model. In *Real-Time Systems Symposium (RTSS), 2010 IEEE 31st*, pages 173–182. IEEE, 2010. (Cited on pages 12, 106, and 122.)

[14] Sanjoy Baruah, Deji Chen, Sergey Gorinsky, and Aloysius Mok. Generalized multiframe tasks. *Real-Time Systems*, 17(1):5–22, 1999. (Cited on pages 12, 106, and 122.)

[15] Sanjoy K Baruah. Feasibility analysis of recurring branching tasks. In *Real-Time Systems, 1998. Proceedings. 10th Euromicro Workshop on*, pages 138–145. IEEE, 1998. (Cited on pages 12, 106, and 122.)

[16] Sanjoy K Baruah. A general model for recurring real-time tasks. In *Real-Time Systems Symposium, 1998. Proceedings. The 19th IEEE*, pages 114–122. IEEE, 1998. (Cited on pages 12, 106, and 122.)

[17] Sanjoy K Baruah. Dynamic-and static-priority scheduling of recurring real-time tasks. *Real-Time Systems*, 24(1):93–128, 2003. (Cited on pages 12 and 122.)

[18] Marko Bertogna and Sanjoy Baruah. Limited preemption edf scheduling of sporadic task systems. *IEEE Transactions on Industrial Informatics*, 6(4):579–591, 2010. (Cited on page 19.)

[19] Marko Bertogna and Michele Cirinei. Response-time analysis for globally scheduled symmetric multiprocessor platforms. In *28th IEEE International Real-Time Systems Symposium (RTSS 2007)*, pages 149–160. IEEE, 2007. (Cited on pages 32 and 38.)

[20] B. Blackham, Y. Shi, S. Chattopadhyay, A. Roychoudhury, and G. Heiser. Timing analysis of a protected operating system kernel. In *2011 IEEE 32nd Real-Time Systems Symposium (RTSS)*, pages 339–348, Nov 2011. (Cited on page 58.)

[21] Konstantinos Bletsas, Neil C. Audsley, Wen-Hung Huang, Jian-Jia Chen, and Geoffrey Nelissen. Errata for three papers (2004-05) on fixed-priority scheduling with self-suspensions. *LITES*, 5(1):02:1–02:20, 2018. (Cited on pages 3 and 61.)

[22] Anne Bouillard and Éric Thierry. An algorithmic toolbox for network calculus. *Discrete Event Dynamic Systems*, 18(1):3–49, Mar 2008. (Cited on page 105.)

[23] Marc Boyer and David Doose. Combining network calculus and scheduling theory to improve delay bounds. In *Proceedings of the 20th International Conference on Real-Time and Network Systems*, pages 51–60, 2012. (Cited on page 28.)

[24] Marc Boyer and Pierre Roux. Embedding network calculus and event stream theory in a common model. In *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–8. IEEE, 2016. (Cited on page 28.)

[25] Reinder J Bril, Johan J Lukkien, Rob I Davis, and Alan Burns. Message response time analysis for ideal controller area network (can) refuted. *proc. of the 5th Int. Work. on Real-Time Net.(RTN'06)*, pages 5–10, 2006. (Cited on pages 3 and 23.)

[26] Alan Burns. *Preemptive priority based scheduling: An appropriate engineering approach*. Citeseer, 1993. (Cited on page 18.)

[27] Giorgio C Buttazzo, Enrico Bini, and Darren Buttle. Rate-adaptive tasks: Model, analysis, and design issues. In *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6. IEEE, 2014. (Cited on page 10.)

[28] Lennart Casparsson, Antal Rajnak, Ken Tindell, and Peter Malmberg. Volcano-a revolution in on-board communications. *Volvo technology report*, 1:9–19, 1998. (Cited on page 4.)

[29] F. Cerqueira, F. Stutz, and B. B. Brandenburg. Prosa: A case for readable mechanized schedulability analysis. In *2016 28th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 273–284, July 2016. (Cited on pages 32, 38, 43, and 119.)

[30] Felipe Cerqueira, Geoffrey Nelissen, and Björn B. Brandenburg. On Strong and Weak Sustainability, with an Application to Self-Suspending Real-Time Tasks. In *30th Euromicro Conference on Real-Time Systems (ECRTS 2018)*, volume 106 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 26:1–26:21, 2018. (Cited on pages 39, 119, and 121.)

[31] Felipe Cerqueira, Felix Stutz, and Björn B Brandenburg. Prosa: A case for readable mechanized schedulability analysis. In *Real-Time Systems (ECRTS), 2016 28th Euromicro Conference on*, pages 273–284. IEEE, 2016. (Cited on page 33.)

[32] The Coq Certifier of CAN Analysis Results. https://team.inria.fr/spades/certican-plus/. (Cited on pages 63, 68, and 94.)

[33] Cheng-Shang Chang. On deterministic traffic regulation and service guarantees: a systematic approach by filtering. *IEEE Transactions on Information Theory*, 44(3):1097–1110, 1998. (Cited on page 27.)

[34] Cheng-Shang Chang. *Performance Guarantees in Communication Networks*. Springer-Verlag, Berlin, Heidelberg, 2000. (Cited on page 27.)

[35] The Coq formalization of a TDMA RTA. https://team.inria.fr/spades/coq-proofs-tdma/. (Cited on page 15.)

[36] James C Corbett. Modeling and analysis of real-time ada tasking programs. In *RTSS*, volume 94, pages 132–141, 1994. (Cited on page 29.)

[37] Mikel Cordovilla, Frédéric Boniol, Eric Noulard, and Claire Pagetti. Multiprocessor schedulability analyser. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, pages 735–741, 2011. (Cited on pages 29 and 57.)

[38] David Costanzo, Zhong Shao, and Ronghui Gu. End-to-end verification of information-flow security for C and assembly programs. In *Proceedings of*

the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), pages 648–664, 2016. (Cited on pages 31, 44, 45, and 58.)

[39] R. L. Cruz. A calculus for network delay. i. network elements in isolation. *IEEE Transactions on Information Theory*, 37(1):114–131, Jan 1991. (Cited on page 27.)

[40] Rene L. Cruz. Quality of service guarantees in virtual circuit switched networks. *IEEE Journal on Selected areas in Communications*, 13(6):1048–1056, 1995. (Cited on page 27.)

[41] Rene L Cruz. Sced+: Efficient management of quality of service guarantees. In *Proceedings. IEEE INFOCOM'98, the Conference on Computer Communications. Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies. Gateway to the 21st Century (Cat. No. 98*, volume 2, pages 625–634. IEEE, 1998. (Cited on page 27.)

[42] Hugo Daigmorte and Marc Boyer. Traversal time for weakly synchronized can bus. In *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, pages 35–44, 2016. (Cited on page 28.)

[43] Robert I. Davis, Alan Burns, Reinder J. Bril, and Johan J. Lukkien. Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised. *Real-Time Systems*, 35(3):239–272, 2007. (Cited on pages 3, 23, 25, 61, and 64.)

[44] Robert I Davis, Liliana Cucu-Grosjean, Marko Bertogna, and Alan Burns. A review of priority assignment in real-time systems. *Journal of systems architecture*, 65:64–82, 2016. (Cited on page 16.)

[45] Robert Ian Davis and Liliana Cucu-Grosjean. A survey of probabilistic schedulability analysis techniques for real-time systems. *LITES: Leibniz Transactions on Embedded Systems*, pages 1–53, 2019. (Cited on page 27.)

[46] Daniel De Rauglaudre. *Vérification formelle de conditions d'ordonnancabilité de tâches temps réel périodiques strictes*, 2012. (Cited on page 32.)

[47] Michael L. Dertouzos. Control robotics: The procedural control of physical processes. In Jack L. Rosenfeld, editor, *Information Processing, Proceedings of the 6th IFIP Congress 1974, Stockholm, Sweden, August 5-10, 1974*, pages 807–813. North-Holland, 1974. (Cited on page 21.)

[48] Raymond Devillers and Joël Goossens. Liu and layland's schedulability test revisited. *Information Processing Letters*, 73(5-6):157–161, 2000. (Cited on page 3.)

[49] José Luis Díaz, Daniel F García, Kanghee Kim, Chang-Gun Lee, L Lo Bello, José María López, Sang Lyul Min, and Orazio Mirabella. Stochastic analysis of periodic real-time systems. In *23rd IEEE Real-Time Systems Symposium, 2002. RTSS 2002.*, pages 289–300. IEEE, 2002. (Cited on page 27.)

[50] Bruno Dutertre. The priority ceiling protocol: formalization and analysis using pvs. In *Proceedings of the 21st IEEE Conference on Real-Time Systems Symposium (RTSS)*, pages 151–160, 1999. (Cited on pages 31 and 57.)

[51] Bruno Dutertre and Victoria Stavridou. Formal analysis for real-time scheduling. In *19th DASC. 19th Digital Avionics Systems Conference. Proceedings (Cat. No. 00CH37126)*, volume 1, pages 1D4–1. IEEE, 2000. (Cited on page 32.)

[52] The eChronos RTOS. https://ts.data61.csiro.au/projects/TS/echronos/. (Cited on page 31.)

[53] The Ergo theorem prover. http://staff.itee.uq.edu.au/pjr/HomePages/ErgoHome.html. (Cited on page 31.)

[54] Elena Fersman, Leonid Mokrushin, Paul Pettersson, and Wang Yi. Schedulability analysis of fixed-priority systems using timed automata. *Theoretical Computer Science*, 354(2):301–317, 2006. (Cited on pages 29 and 57.)

[55] Elena Fersman, Paul Pettersson, and Wang Yi. Timed automata with asynchronous processes: Schedulability and decidability. In Joost-Pieter Katoen and Perdita Stevens, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 67–82, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg. (Cited on page 29.)

[56] Colin Fidge, Peter Kearney, and Mark Utting. *Formal specification and interactive proof of a simple real-time scheduler*, 1994. (Cited on page 30.)

[57] Norman Finn. Introduction to time-sensitive networking. *IEEE Communications Standards Magazine*, 2(2):22–28, 2018. (Cited on page 19.)

[58] Pascal Fradet, Xiaojie Guo, Jean-François Monin, and Sophie Quinton. A generalized digraph model for expressing dependencies. In *Proceedings of the*

26th International Conference on Real-Time Networks and Systems, pages 72–82, 2018. (Cited on pages 5 and 66.)

[59] Pascal Fradet, Xiaojie Guo, Jean-François Monin, and Sophie Quinton. Certican: A tool for the coq certification of can analysis results. In *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 182–191. IEEE, 2019. (Cited on pages 5 and 38.)

[60] Pascal Fradet, Maxime Lesourd, Jean-François Monin, and Sophie Quinton. A generic coq proof of typical worst-case analysis. In *2018 IEEE Real-Time Systems Symposium (RTSS)*, pages 218–229. IEEE, 2018. (Cited on page 39.)

[61] Goran Frehse, Arne Hamann, Sophie Quinton, and Matthias Woehrle. Formal analysis of timing effects on closed-loop properties of control software. In *2014 IEEE Real-Time Systems Symposium*, pages 53–62. IEEE, 2014. (Cited on page 27.)

[62] Laurent George, Nicolas Rivierre, and Marco Spuri. Preemptive and non-preemptive real-time uniprocessor scheduling. *Technique Report*, 1996. (Cited on page 25.)

[63] Georges Gonthier, Andrea Asperti, Jeremy Avigad, Yves Bertot, Cyril Cohen, François Garillot, Stéphane Le Roux, Assia Mahboubi, Russell O'Connor, Sidi Ould Biha, Ioana Pasca, Laurence Rideau, Alexey Solovyev, Enrico Tassi, and Laurent Théry. A Machine-Checked Proof of the Odd Order Theorem. In Sandrine Blazy, Christine Paulin, and David Pichardie, editors, *ITP 2013, 4th Conference on Interactive Theorem Proving*, volume 7998 of *LNCS*, pages 163–179, Rennes, France, July 2013. Springer. (Cited on pages 30 and 33.)

[64] Michael Gonzalez, Harbour Mark, H Klein, and John P Lehoczky. Fixed priority scheduling of periodic tasks with varying execution priority. In *In Proceedings, IEEE Real-Time Systems Symposium*. Citeseer, 1991. (Cited on page 24.)

[65] Klaus Gresser. An event model for deadline verification of hard real-time systems. In *Fifth Euromicro Workshop on Real-Time Systems*, pages 118–123. IEEE, 1993. (Cited on page 27.)

[66] Ronghui Gu, Jérémie Koenig, Tahina Ramananandro, Zhong Shao, Xiongnan (Newman) Wu, Shu-Chun Weng, Haozhong Zhang, and Yu Guo. Deep specifications and certified abstraction layers. In *Proceedings of the 42nd*

Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL), pages 595–608, 2015. (Cited on pages 4, 30, 31, 40, 44, 45, and 58.)

[67] Ronghui Gu, Zhong Shao, Hao Chen, Xiongnan (Newman) Wu, Jieung Kim, Vilhelm Sjöberg, and David Costanzo. Certikos: An extensible architecture for building certified concurrent os kernels. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 653–669. USENIX Association, 2016. (Cited on pages 43 and 45.)

[68] Ronghui Gu, Zhong Shao, Jieung Kim, Xiongnan (Newman) Wu, Jérémie Koenig, Vilhelm Sjöberg, Hao Chen, David Costanzo, and Tahina Ramananandro. Certified concurrent abstraction layers. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pages 646–661, 2018. (Cited on page 45.)

[69] N. Guan, C. Gu, M. Stigge, Q. Deng, and W. Yi. Approximate response time analysis of real-time task graphs. In *2014 IEEE Real-Time Systems Symposium*, pages 304–313, Dec 2014. (Cited on page 112.)

[70] Nan Guan, Zonghua Gu, Qingxu Deng, Shuaihong Gao, and Ge Yu. Exact schedulability analysis for static-priority global multiprocessor scheduling using model-checking. In *IFIP International Workshop on Software Technolgies for Embedded and Ubiquitous Systems*, pages 263–272, 2007. (Cited on pages 29, 43, and 57.)

[71] Xiaojie Guo, Maxime Lesourd, Mengqi Liu, Lionel Rieg, and Zhong Shao. Integrating formal schedulability analysis into a verified os kernel. In *International Conference on Computer Aided Verification*, pages 496–514. Springer, 2019. (Cited on pages 5 and 44.)

[72] Xiaojie Guo, Sophie Quinton, Pascal Fradet, and Jean-François Monin. Work-in-progress: Toward a coq-certified tool for the schedulability analysis of tasks with offsets. In *2017 IEEE Real-Time Systems Symposium (RTSS)*, pages 387–389. IEEE, 2017. (Cited on page 38.)

[73] The High-Assurance Cyber Military Systems project. http://ts.data61.csiro.au/projects/TS/SMACCM/. (Cited on page 40.)

[74] Thomas A Henzinger. The theory of hybrid automata. In *Verification of digital and hybrid systems*, pages 265–292. Springer, 2000. (Cited on page 29.)

[75] Chao Huang, Wenchao Li, and Qi Zhu. Formal verification of weakly-hard systems. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, pages 197–207, 2019. (Cited on page 29.)

[76] Jean-Bernard Itier. A380 integrated modular avionics—the history, objectives and challenges of the deployment of ima on a380. In *Proceedings of the ARTIST2 Meeting on Integrated Modular Avionics, Roma, Italy*, pages 12–13, 2007. (Cited on page 3.)

[77] Matthias Ivers and Rolf Ernst. Probabilistic network loads with dependencies and the effect on queue sojourn times. In *International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness*, pages 280–296. Springer, 2009. (Cited on page 27.)

[78] Mathai Joseph and Paritosh Pandya. Finding response times in a real-time system. *The Computer Journal*, 29(5):390–395, 1986. (Cited on page 22.)

[79] Jin Hyun Kim, Kim G Larsen, Brian Nielsen, Marius Mikučionis, and Petur Olsen. Formal analysis and testing of real-time automotive systems using uppaal tools. In *International Workshop on Formal Methods for Industrial Critical Systems*, pages 47–61. Springer, 2015. (Cited on page 29.)

[80] Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. seL4: formal verification of an OS kernel. In *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles (SOSP)*, pages 207–220, 2009. (Cited on pages 4, 40, 43, and 44.)

[81] Gerwin Klein, Ralf Huuck, and Bastian Schlich. Operating system verification. *Journal of Automated Reasoning*, 42(2-4):123–124, 2009. (Cited on page 58.)

[82] Leonie Köhler, Borislav Nikolic, Rolf Ernst, and Marc Boyer. Increasing accuracy of timing models: From cpa to cpa+. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1210–1215. IEEE, 2019. (Cited on page 28.)

[83] Jan Korst, Emile Aarts, Jan Karel Lenstra, and Jaap Wessels. Periodic multiprocessor scheduling. In *PARLE'91 Parallel Architectures and Languages Europe*, pages 166–178. Springer, 1991. (Cited on page 32.)

[84] J.H.M. Korst. *Periodic multiprocessor scheduling*. PhD thesis, Department of Mathematics and Computer Science, 1992. (Cited on page 32.)

[85] Jean J. Labrosse. *Microc/OS-II*. R & D Books, 2nd edition, 1998. (Cited on page 58.)

[86] Phillip A Laplante and Seppo J Ovaska. *Real-time systems design and analysis: tools for the practitioner*. John Wiley and Sons, 2011. (Cited on page 32.)

[87] Kim G Larsen, Paul Pettersson, and Wang Yi. Uppaal in a nutshell. *International journal on software tools for technology transfer*, 1(1-2):134–152, 1997. (Cited on page 29.)

[88] J-Y Le Boudec. Application of network calculus to guaranteed service networks. *IEEE Transactions on Information theory*, 44(3):1087–1096, 1998. (Cited on page 27.)

[89] Jean-Yves Le Boudec and Patrick Thiran. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*, 06 2004. (Cited on page 27.)

[90] John P Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *[1990] Proceedings 11th Real-Time Systems Symposium*, pages 201–209. IEEE, 1990. (Cited on pages 21 and 24.)

[91] Xavier Leroy. Formal verification of a realistic compiler. *Communications of the ACM*, 52(7):107–115, 2009. (Cited on pages 4 and 30.)

[92] Xavier Leroy. A formally verified compiler back-end. *Journal of Automated Reasoning*, 43(4):363, 2009. (Cited on pages 33 and 40.)

[93] Joseph Y-T Leung and Jennifer Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance evaluation*, 2(4):237–250, 1982. (Cited on page 21.)

[94] Qing Li and Caroline Yao. *Real-time concepts for embedded systems*. CRC Press, 2003. (Cited on page 32.)

[95] Didier Lime et al. Formal verification of real-time systems with preemptive scheduling. *Real-Time Systems*, 41(2):118–151, 2009. (Cited on page 30.)

[96] Chung Laung Liu and James W Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)*, 20(1):46–61, 1973. (Cited on pages 3, 10, 21, 22, and 121.)

[97] Jane W. S. W. Liu. *Real-Time Systems*. Prentice Hall PTR, USA, 1st edition, 2000. (Cited on pages 31 and 32.)

[98] Mengqi Liu, Lionel Rieg, Zhong Shao, Ronghui Gu, David Costanzo, Jung-Eun Kim, and Man-Ki Yoon. Compositional verification of preemptive OS kernels with temporal and spatial isolation. Technical Report YALEU/DCS/TR-1549, Dept. of Computer Science, Yale University, 2019. (Cited on pages 44, 45, 46, and 58.)

[99] Mengqi Liu, Lionel Rieg, Zhong Shao, Ronghui Gu, David Costanzo, Jung-Eun Kim, and Man-Ki Yoon. Virtual timeline: a formal abstraction for verifying preemptive schedulers with temporal isolation. *Proceedings of the ACM on Programming Languages*, 4(POPL):1–31, 2019. (Cited on page 31.)

[100] Etienne Mabille, Marc Boyer, Loïc Fejoz, and Stephan Merz. Towards certifying network calculus. In *International Conference on Interactive Theorem Proving*, pages 484–489. Springer, 2013. (Cited on page 40.)

[101] P. Marti, R. Villa, J. M. Fuertes, and G. Fohle. On real-time control tasks schedulability. In *2001 European Control Conference (ECC)*, pages 2227–2232, Sep. 2001. (Cited on page 10.)

[102] Dorin Maxim and Liliana Cucu-Grosjean. Response time analysis for fixed-priority tasks with multiple probabilistic parameters. In *2013 IEEE 34th Real-Time Systems Symposium*, pages 224–235. IEEE, 2013. (Cited on page 27.)

[103] The MicroC/OS-II RTOS. https://www.micrium.com/books/ucosii/. (Cited on page 31.)

[104] Morteza Mohaqeqi, Jakaria Abdullah, Nan Guan, and Wang Yi. Schedulability analysis of synchronous digraph real-time tasks. In *Real-Time Systems (ECRTS), 2016 28th Euromicro Conference on*, pages 176–186. IEEE, 2016. (Cited on pages 13, 106, and 122.)

[105] A. K. Mok and D. Chen. A multiframe model for real-time tasks. In *17th IEEE Real-Time Systems Symposium*, pages 22–29, Dec 1996. (Cited on pages 11, 106, and 121.)

[106] Aloysius Ka-Lau Mok. *Fundamental design problems of distributed systems for the hard-real-time environment*. PhD thesis, Massachusetts Institute of Technology, 1983. (Cited on pages 10 and 121.)

[107] Aurélien Monot, Nicolas Navet, Bernard Bavoux, and Cristian Maxim. Fine-grained simulation in the design of automotive communication systems. In *ERTSS-Embedded Real Time Software and Systems-2012*, 2012. (Cited on page 93.)

[108] Luke Nelson, Helgi Sigurbjarnarson, Kaiyuan Zhang, Dylan Johnson, James Bornholt, Emina Torlak, and Xi Wang. Hyperkernel: Push-button verification of an OS kernel. In *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP), Shanghai, China, October 28-31, 2017*, pages 252–269, 2017. (Cited on page 43.)

[109] William R Nichols and Sarah Sheard. Faa research project on system complexity effects on aircraft safety: Candidate complexity metrics. *White paper, Software Engineering Institute, Carnegie Mellon University*, 2016. (Cited on page 3.)

[110] RTaW-Pegase: a Tool for Modeling, Simulation and automated Configuration of communication networks. http://www.realtimeatwork.com/software/rtaw-pegase/. (Cited on pages 5 and 61.)

[111] Chao Peng and Haibo Zeng. Response time analysis of digraph real-time tasks scheduled with static priority: generalization, approximation, and improvement. *Real-Time Systems*, 54(1):91–131, Jan 2018. (Cited on page 125.)

[112] A Library for formally proven schedulability analysis. http://prosa.mpi-sws.org/. (Cited on pages 4, 32, 33, 68, and 94.)

[113] Sophie Quinton, Torsten T. Bone, Julien Hennig, Moritz Neukirchner, Mircea Negrean, and Rolf Ernst. Typical worst case response-time analysis and its use in automotive network design. In *The 51st Annual Design Automation Conference 2014, DAC '14, San Francisco, CA, USA, June 1-5, 2014*, pages 44:1–44:6, 2014. (Cited on page 93.)

[114] Sophie Quinton, Matthias Hanke, and Rolf Ernst. Formal analysis of sporadic overload in real-time systems. In *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 515–520. IEEE, 2012. (Cited on page 26.)

[115] Ragunathan Rajkumar. *Synchronization in real-time systems: a priority inheritance approach*, volume 151. Springer Science & Business Media, 2012. (Cited on page 32.)

[116] Lucien Rakotomalala, Marc Boyer, and Pierre Roux. Formal verification of real-time networks. In *JRWRTC 2019, Junior Workshop RTNS 2019*, 2019. (Cited on page 39.)

[117] The Design and Analysis of Reconfigurable Multi-view Embedded Systems Project. https://project.inria.fr/caserm/. (Cited on page 33.)

[118] Kai Richter. *Compositional Scheduling Analysis Using Standard Event Models*, Dec 2004. (Cited on page 27.)

[119] Silvain Rideau and Xavier Leroy. Validating register allocation and spilling. In *International Conference on Compiler Construction*, pages 224–243. Springer, 2010. (Cited on page 40.)

[120] Jonas Rox and Rolf Ernst. Compositional performance analysis with improved analysis techniques for obtaining viable end-to-end latencies in distributed embedded systems. *International Journal on Software Tools for Technology Transfer*, 15(3):171–187, 2013. (Cited on page 27.)

[121] Hanrijanto Sariowan, Rene L Cruz, and George C Polyzos. Scheduling for quality of service guarantees via service curves. In *Proceedings of Fourth International Conference on Computer Communications and Networks-IC3N'95*, pages 512–520. IEEE, 1995. (Cited on page 27.)

[122] Simon Schliecker. *Performance analysis of multiprocessor real-time systems with shared resources*. Institut für Datentechnik und Kommunikationsnetze, Braunschweig, 2011. Zugleich: Braunschweig, Techn. Univ., Diss., 2011. (Cited on page 27.)

[123] Thomas Sewell, Felix Kam, and Gernot Heiser. High-assurance timing analysis for a high-assurance real-time operating system. *Real-Time Syst.*, 53(5):812–853, September 2017. (Cited on page 58.)

[124] Lui Sha, Ragunathan Rajkumar, and John P Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Transactions on computers*, 39(9):1175–1185, 1990. (Cited on pages 31 and 32.)

[125] Abraham Silberschatz, Peter Baer Galvin, and Greg Gagne. *Operating system concepts essentials*. John Wiley & Sons, Inc., 2014. (Cited on page 32.)

[126] Purnendu Sinha and Neeraj Suri. On the use of formal techniques for analyzing dependable real-time protocols. In *Proceedings 20th IEEE Real-Time Systems*

Symposium (Cat. No. 99CB37054), pages 126–135. IEEE, 1999. (Cited on page 31.)

[127] CAN Specification. Version 2.0. *Robert Bosch GmbH*, 1991. (Cited on pages 61 and 63.)

[128] M. Stigge and W. Yi. Hardness results for static priority real-time scheduling. In *2012 24th Euromicro Conference on Real-Time Systems*, pages 189–198, July 2012. (Cited on page 121.)

[129] Martin Stigge, Pontus Ekberg, Nan Guan, and Wang Yi. The digraph real-time task model. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2011 17th IEEE*, pages 71–80. IEEE, 2011. (Cited on pages 12, 97, 102, 106, and 122.)

[130] Martin Stigge, Pontus Ekberg, Nan Guan, and Wang Yi. On the tractability of digraph-based task models. In *Real-Time Systems (ECRTS), 2011 23rd Euromicro Conference on*, pages 162–171. IEEE, 2011. (Cited on page 122.)

[131] Martin Stigge, Nan Guan, and Wang Yi. Refinement-based exact response-time analysis. In *26th Euromicro Conference on Real-Time Systems, ECRTS 2014, Madrid, Spain, July 8-11, 2014*, pages 143–152, 2014. (Cited on page 85.)

[132] Martin Stigge and Wang Yi. Combinatorial abstraction refinement for feasibility analysis of static priorities. *Real-Time Systems*, 51(6):639–674, 2015. (Cited on pages 85 and 125.)

[133] Martin Stigge and Wang Yi. Graph-based models for real-time workload: a survey. *Real-time systems*, 51(5):602–636, 2015. (Cited on page 11.)

[134] Youcheng Sun and Giuseppe Lipari. A pre-order relation for exact schedulability test of sporadic tasks on multiprocessor global fixed-priority scheduling. *Real-Time Systems*, 52(3):323–355, 2016. (Cited on pages 29 and 30.)

[135] SymTA/S: Model-based timing analysis and optimization. https://auto.luxoft.com/uth/timing-analysis-tools/. (Cited on page 5.)

[136] The Coq Development Team. *The Coq Proof Assistant Reference Manual*. INRIA, 8.4pl4 edition, 2014. (Cited on page 45.)

[137] The Fomal Proofs for Real-Time Systems Project. http://rt-proofs.inria.fr/. (Cited on page 33.)

[138] The Coq proof assistant. http://coq.inria.fr. (Cited on pages 30 and 68.)

[139] The Isabelle/HOL proof assistant. https://isabelle.in.tum.de/. (Cited on page 30.)

[140] The PVS proof assistant. https://pvs.csl.sri.com/. (Cited on page 30.)

[141] Lothar Thiele, Samarjit Chakraborty, and Martin Naedele. Real-time calculus for scheduling hard real-time systems. In *Circuits and Systems, 2000. Proceedings. ISCAS 2000 Geneva. The 2000 IEEE International Symposium on*, volume 4, pages 101–104. IEEE, 2000. (Cited on pages 11, 27, 105, and 121.)

[142] Ken Tindell. *Using offset information to analyse static priority pre-emptively scheduled task sets*. Technical report YCS 182. University of York, Department of Computer Science, 1992. (Cited on pages 62, 73, 97, and 106.)

[143] Ken Tindell. *Adding time-offsets to schedulability analysis*. University of York, Department of Computer Science, 1994. (Cited on pages 62 and 112.)

[144] Ken Tindell and Alan Burns. Guaranteeing message latencies on controller area network (CAN). In *Proceedings of 1st international CAN conference*, pages 1–11, 1994. (Cited on pages 3, 43, and 61.)

[145] Ken Tindell, Alan Burns, and Andy Wellings. Calculating controller area network (CAN) message response times. *Control Engineering Practice*, 3(8):1163–1169, 1995. (Cited on pages 3, 43, and 61.)

[146] Ken Tindell and John Clark. Holistic schedulability analysis for distributed hard real-time systems. *Microprocessing and microprogramming*, 40(2-3):117–134, 1994. (Cited on page 27.)

[147] Ken Tindell, H. Hanssmon, and Andy J. Wellings. Analysing real-time communications: Controller area network (CAN). In *Proceedings of the 15th IEEE Real-Time Systems Symposium (RTSS), San Juan, Puerto Rico, December 7-9, 1994*, pages 259–263, 1994. (Cited on pages 3, 23, 43, and 61.)

[148] Ronald M Tol. A small real-time kernel proven correct. In *[1992] Proceedings Real-Time Systems Symposium*, pages 227–230. IEEE, 1992. (Cited on page 31.)

[149] Mark Utting, Peter Robinson, and Ray Nickson. Ergo 6: A generic proof engine that uses prolog proof technology. *LMS Journal of Computation and Mathematics*, 5:194–219, 2002. (Cited on page 31.)

[150] Nikhil Kumar Varma. *Formal analysis of fault tolerant real time multiprocessor allocation and scheduling protocols*. PhD thesis, Concordia University, 2004. (Cited on page 31.)

[151] Yun Wang and Manas Saksena. Scheduling fixed-priority tasks with preemption threshold. In *Proceedings Sixth International Conference on Real-Time Computing Systems and Applications. RTCSA'99 (Cat. No. PR00306)*, pages 328–335. IEEE, 1999. (Cited on page 19.)

[152] Matthew Wilding. A machine-checked proof of the optimality of a real-time scheduling policy. In *Proceedings of the 10th International Conference on Computer Aided Verification (CAV)*, pages 369–378, 1998. (Cited on pages 31 and 57.)

[153] Matthew M Wilding, David S Hardin, and David A Greve. Invariant performance: A statement of task isolation useful for embedded application integration. In *Dependable Computing for Critical Applications 7*, pages 287–300. IEEE, 1999. (Cited on page 31.)

[154] Matthew Michael Wilding. *Machine-checked real-time system verification*, 1996. (Cited on page 31.)

[155] Fengwei Xu, Ming Fu, Xinyu Feng, Xiaoran Zhang, Hui Zhang, and Zhaohui Li. A practical verification framework for preemptive OS kernels. In Swarat Chaudhuri and Azadeh Farzan, editors, *Proceedings Computer Aided Verification (CAV), Part II*, pages 59–79. Springer International Publishing, 2016. (Cited on pages 31, 44, and 58.)

[156] Qiwen Xu and Naijun Zhan. Formalising scheduling theories in duration calculus. *Nord. J. Comput.*, 14(3):173–201, 2008. (Cited on page 57.)

[157] Wenbo Xu, Zain AH Hammadeh, Alexander Kröller, Rolf Ernst, and Sophie Quinton. Improved deadline miss models for real-time systems using typical worst-case analysis. In *2015 27th Euromicro Conference on Real-Time Systems*, pages 247–256. IEEE, 2015. (Cited on page 26.)

[158] Victor Yodaiken. Against priority inheritance, 2004. (Cited on page 32.)

[159] Patrick Meumeu Yomsi, Dominique Bertrand, Nicolas Navet, and Robert I Davis. Controller area network (can): Response time analysis with offsets. In *2012 9th IEEE International Workshop on Factory Communication Systems*, pages 43–52. IEEE, 2012. (Cited on pages 62 and 63.)

[160] Zheng Yuhua and Zhou Chaochen. A formal proof of the deadline driven scheduler. In *International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 756–775, 1994. (Cited on page 57.)

[161] Nan Zhang, Zhenhua Duan, Cong Tian, and Dingzhu Du. A formal proof of the deadline driven scheduler in pptl axiomatic system. *Theoretical Computer Science*, 554:229–253, 2014. (Cited on page 31.)

[162] Xingyuan Zhang, Christian Urban, and Chunhan Wu. Priority inheritance protocol proved correct. In Lennart Beringer and Amy Felty, editors, *Interactive Theorem Proving*, pages 217–232, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. (Cited on page 32.)

[163] Xingyuan Zhang, Christian Urban, and Chunhan Wu. Priority inheritance protocol proved correct. *Journal of Automated Reasoning*, 64(1):73–95, 2020. (Cited on page 32.)