From Intuition to Coq: A Case Study in Verified Response-Time Analysis of FIFO Scheduling

Kimaya Bedarkar Mariam Vardishvili Sergey Bozhko Marco Maida Björn B. Brandenburg
Max Planck Institute for Software Systems (MPI-SWS)

***Abstract*—Response-time analysis (RTA) is a key technique for the analysis of (not only) safety-critical real-time systems. It is hence crucial for published RTAs to be *safe* (*i.e.*, correct), but historically this has not always been the case. To ensure the trustworthiness of RTAs, recent work has pioneered the use of formal verification. The PROSA open-source project, in particular, relies on the COQ proof assistant to mechanically check all proofs. While highly effective at eradicating human error, such formalization and automatic validation of mathematical reasoning still faces barriers to more widespread adoption as most researchers active today are not yet accustomed to the use of proof assistants. To make this approach more broadly accessible, this paper presents a case study in the verification of a novel RTA for sporadic tasks under FIFO scheduling using the COQ proof assistant. The RTA is derived twice, first using traditional, intuition-based reasoning, and once more formally in a style that highlights the similarity to the intuitive argument. The verified RTA is of interest in itself: experiments with synthetic workloads based on an automotive benchmark show the new RTA to clearly outperform a prior RTA for FIFO scheduling. The paper further explores the performance of FIFO scheduling relative to traditional fixed-priority and earliest-deadline-first approaches, showing that FIFO scheduling can benefit lower-rate tasks.**

I. INTRODUCTION

The defining characteristic of real-time systems is that their constituent processes must meet timing constraints. Especially in safety-critical systems, such timing constraints are typically strict and subject to *a priori* validation. A common approach to this end is the use of a *response-time analysis* (RTA) [30, 33] to bound the worst-case response times of time-critical processes.

RTAs have been developed for a large variety of scheduling policies and workload characteristics. Published RTAs differ in many details large and small, but they all share one trait: it is inherent in their intended use that they must be *sound*. The reason is obvious—if a flawed RTA is used to analyze a safety-critical system, an incorrect (*i.e.*, optimistic) response-time "bound" could undermine the desired safety assurances.

Alas, there is ample evidence that the literature on real-time systems, despite all efforts and critical peer-review, has harbored a nontrivial number of flawed analyses over the years (*e.g.*, [14, 18, 28, 32]), with new refutations still continuing to appear on a regular basis (*e.g.*, [29, 53]). To counteract this regrettable trend, recent work has started using formal verification methods, in particular the COQ proof assistant [1], to minimize the risk of human error in timing analyses intended for critical systems [9, 10, 12, 13, 23–25, 28, 40, 47].

As we argue in more detail in Sec. II-C, this is a necessary and promising reaction. While formal, machine-checked proofs arguably should have been adopted decades ago [21, 51], it is only recently that advances in tooling [1], libraries [39], and educational materials [46] have made such an approach viable at broader scale and for a broader audience.

Even then, formal verification still comes with a nontrivial learning curve. Its widespread adoption is further hindered by the fact that many researchers active today have little to no experience in using proof assistants. The situation is also not helped by the observation that few, if any, of the above-cited papers are aimed at novice users not yet accustomed to formalized mathematics and machine-checked proofs.

Motivated by these circumstances, and in the hope of furthering the use of proof assistants in the analysis of real-time systems, we present a case study in the verification of a novel RTA for *first-in-first-out* (FIFO) scheduling. To provide an easy-to-follow, self-contained example, we derive the proposed RTA twice, first in a traditional style rooted in intuitive reasoning, and then formally such that its correctness argument can be checked with the COQ proof assistant.

Why FIFO? Under FIFO scheduling, or, equivalently, *first-come-first-served* (FCFS) scheduling, jobs are executed simply in order of their arrival times, disregarding any differences in priority or urgency. Compared to the *fixed-priority* (FP) or *earliest-deadline-first* (EDF) policies, FIFO cannot match their hard-real-time schedulability [11, 17]. Nonetheless, FIFO scheduling is a common choice in the design of simple, low-cost embedded systems and device drivers, in model-based design [43, 44], in the Linux kernel [7], and real-time packet-switching and processing [22, 37].

The key reason to adopt FIFO is its *simplicity*, which manifests in various aspects. FIFO scheduling is **(1)** straightforward to implement, given that it only requires the management of a list of jobs. Since jobs are executed in arrival order, **(2)** no preemption occurs, greatly reducing the necessary bookkeeping code in the OS, context-switch overheads, cache-related delays, and the total state space of the system.

Moreover, **(3)** FIFO has remarkably low scheduling overheads, as a scheduling decision happens only once per job, and the only operation performed—extracting the head element from a FIFO queue—takes negligible time. In addition, the scheduler can remain completely loop-free and thus **(4)** takes constant time. In contrast, obtaining a constant-time scheduler is generally not possible with EDF scheduling, while for FP scheduling it requires limiting the number of priorities to a constant number. FIFO's inherent simplicity also makes it **(5)** ideally suited for implementation in hardware. Finally, FIFO **(6)** exhibits no scheduling anomalies w.r.t. execution times on both uni- and multiprocessors [4] and **(7)** is trivially starvation-free (*i.e.*, every job eventually completes, unless intentionally dropped), which ensures bounded tardiness [38].

For these reasons, FIFO finds widespread use in real-time systems, especially in the case of non-periodic workloads (*e.g.*, as commonly found in energy-constrained devices that maximize the time spent in low-power sleep states [3, 52]). It may therefore come as a surprise that the existing real-time literature on this pervasively used policy is remarkably sparse.

Related work. While FIFO scheduling has been extensively explored from a stochastic perspective in queuing theory (*e.g.*, [6, 27, 41]), the policy has received only limited attention by the real-time systems community. An RTA for distributed systems using FIFO was proposed by George and Minet [26] more than two decades ago. A much more recent analysis of uniprocessor FIFO scheduling by Altmeyer et al. [4] focuses mainly on periodic tasks with offsets. Offset tuning, in particular, has been shown to be highly effective for achieving high schedulability under FIFO scheduling if all tasks are periodic [42]. Altmeyer et al. also proposed a simple RTA for sporadic tasks under FIFO scheduling, which however applies only if all tasks have constrained deadlines and no deadlines are missed [4]. Davis et al. proposed a schedulability analysis for CAN messages in a network with both FIFO and FP nodes and studied the detrimental effect of FIFO queuing on hard real-time schedulability [19]. FIFO queuing has also been studied in the analysis of real-time Ethernet variants [20, 49], and can be characterized using network calculus [8]. Finally, for soft real-time multiprocessor systems, Leontyev and Anderson derived tardiness bounds for global FIFO scheduling [38].

This paper. We make three contributions in this paper. First, in Sec. II, we propose a novel uniprocessor RTA for sporadic tasks under FIFO scheduling. The proposed RTA is substantially more flexible than the state of the art, along two dimensions. For one, it applies to tasks with *arbitrary deadlines* [36], rather than just constrained deadlines. Furthermore, all of the above-cited analyses support only well-structured arrival processes described by a scalar parameter, namely a period or job minimum inter-arrival time. In contrast, our new RTA supports *arbitrary arrival curves* [31], which can express a wide range of irregular and bursty arrival processes.

Our second and main contribution, presented in Secs. III–V, is a case study explaining how we verified the proposed RTA with the COQ proof assistant. Our formal proof rests on the semantic foundations provided by the PROSA open-source project [2, 12] and uses its *abstract RTA* (aRTA) framework [10]. We review each of these elements assuming no prior familiarity (Sec. III), detail how we *instantiated* aRTA to obtain the verified RTA (Sec. IV), and then reflect on lessons learned, possible alternatives, and future extensions (Sec. V).

Finally, we report on an empirical evaluation of the proposed RTA in Sec. VI. Experiments on synthetic workloads based on the Bosch automotive benchmark due to Kramer et al. [35] show the new RTA to compare favorably with the state of the art [4]. Additionally, we report on an exploration of the performance of FIFO relative to FP and EDF scheduling.

In conclusion, this paper provides a case study in formalized mathematics in the real-time systems domain, by example of a novel RTA for FIFO scheduling that is of interest in itself.

II. A Response-Time Analysis of FIFO Scheduling

To begin, we introduce our system model and then derive an RTA for FIFO from first principles.

A. System Model

We employ a discrete time model, and let $\mathbb{T} = \mathbb{N}$ denote the time domain and $\varepsilon \triangleq 1$ the indivisible least unit of time.

The workload is a set of n sporadic real-time tasks $\tau \triangleq \{\tau_1, \tau_2, ..., \tau_n\}$. Each task $\tau_i \triangleq (C_i, D_i, \alpha_i)$ has a *worst-case execution time* C_i, a *relative deadline* D_i, and an *arrival-bound function* $\alpha_i(\Delta)$. The role of $\alpha_i(\Delta)$ is to upper-bound the number of activations of τ_i in any time window of length Δ. At each activation, a task *releases* a corresponding *job*, *i.e.*, the job *arrives* and is enqueued for execution. The j-th job of the i-th task is denoted by $J_{i,j}$ and is characterized by its *arrival time* $a_{i,j}$, an *absolute deadline* $d_{i,j} = a_{i,j} + D_i$, and a *cost* $c_{i,j}$ with $0 \leq c_{i,j} \leq C_i$. Jobs do not self-suspend. Arrival times are unknown *a priori*, but comply with $\alpha_i(\Delta)$:

$$\forall t, \forall \Delta, \ |\{J_{i,j} \ | \ t \leq a_{i,j} < t + \Delta\}| \leq \alpha_i(\Delta). \qquad (1)$$

The *request-bound function* $RBF_i(\Delta) \triangleq \alpha_i(\Delta) \cdot C_i$ bounds the total cost of all jobs of τ_i arriving in any interval of length Δ.

We study *ideal unit-speed uniprocessors*, which means that, at any instant, the processor is either *idle* or exactly one job is *scheduled* and receives one unit of processor service. Any overheads are included in each task's cost parameter C_i.

According to the FIFO scheduling policy, a job $J_{h,k}$ has higher-or-equal priority than another job $J_{i,j}$ if $J_{h,k}$ arrives no later than $J_{i,j}$ (*i.e.*, $a_{h,k} \leq a_{i,j}$). Tie-breaking between equal-priority jobs is arbitrary, *i.e.*, the analysis is correct regardless of which tie-breaking policy is used. The schedule is assumed to be *work-conserving*: the processor never idles if a *pending*—*i.e.*, arrived, but yet not completed—job is available.

Since FIFO naturally constrains the processor to complete each scheduled job without preemption, the preemption policy does not affect the response time of jobs. Therefore, we impose no restriction on whether (or at what times) jobs may be preempted. Jobs may further access shared resources such as shared data structures, which we similarly do not explicitly model since FIFO's run-to-completion semantics trivially ensure mutual exclusion.

B. FIFO Response-Time Analysis

With the system model in place, we next introduce a response-time bound for the FIFO scheduling policy with arbitrary deadlines and arbitrary arrival curves. Intuitively, a job's response time under FIFO scheduling is fairly obvious—it is simply the job's own execution time plus that of all still-pending, earlier-arrived jobs. However, the modeling flexibility afforded by arbitrary arrival curves and the possibility that jobs may finish past their deadline make it somewhat less trivial to determine the set of "all still-pending, earlier-arrived jobs," as a large backlog of pending jobs may have built up over time.

Our analysis therefore relies on the *busy-window principle*, a classic concept in real-time scheduling [5, 18, 30, 33, 36] that allows starting the analysis from a moment of zero backlog. Intuitively, the busy window of an arbitrary job $J_{i,j}$ is a time interval (*i.e.*, a *window*) in which the processor continuously executes only higher-or-equal-priority jobs delaying $J_{i,j}$ or $J_{i,j}$ itself (*i.e.*, the processor is *busy*). More precisely, $J_{i.j}$'s busy window is the longest such interval. Clearly, $J_{i,j}$'s busy window ends only after (or with) the completion of $J_{i,j}$.

Now, job $J_{i,j}$'s response time can be bounded as follows. Suppose $J_{i,j}$'s busy window starts at time t_s, and let $A \triangleq a_{i,j} - t_s$ denote $J_{i,j}$'s relative arrival time within its busy window, as illustrated in Fig. 1. As already noted, at every point in $J_{i,j}$'s busy window, $J_{i,j}$ is either itself executing or incurring *interference* due to the execution of jobs that arrived prior to (or together with) $J_{i,j}$. It follows that every interfering job necessarily arrives no earlier than at time t_s and no later than at time $a_{i,j}$, *i.e.*, during the interval $[t_s, t_s + A]$. Therefore, the *interference bound function*

$$\mathit{IBF}(A) = \left(\sum_{\tau_k \in \tau} \mathit{RBF}_k(A + \varepsilon) \right) - C_i \tag{2}$$

upper-bounds the delay incurred by job $J_{i,j}$. Note that $\mathit{IBF}(A)$ includes the task under analysis τ_i in the summation to account for *self-interference* by earlier jobs of the same task. Conversely, since $J_{i,j}$ does not interfere with itself, its cost C_i is subtracted. The "$+\varepsilon$" in Eq. (2) is necessary because the interval under consideration $[t_s, t_s + A]$ is closed, *i.e.*, $|[t_s, t_s + A]| = A + \varepsilon$.

The above bound on interference can be used to calculate a lower bound on the service received by a job in its busy window. In the busy window, consider a time $t > a_{i,j}$ after the job's arrival, and define its offset relative to $a_{i,j}$ as $F \triangleq t - a_{i,j}$. Since the total interference incurred by $J_{i,j}$ during the interval $[t_s, t) = [t_s, t_s + A + F)$ is capped at $\mathit{IBF}(A)$, it follows that $J_{i,j}$ must have received *at least* $A + F - \mathit{IBF}(A)$ units of service by time t (if it did not already complete prior to t). Since $J_{i,j}$ requires at most C_i units of service, solving $C_i = A + F - \mathit{IBF}(A)$ yields a response-time bound: $J_{i,j}$ completes no later than $F = \mathit{IBF}(A) + C_i - A = \left(\sum_{\tau_k \in \tau} \mathit{RBF}_k(A + \varepsilon)\right) - A$ time units after its arrival, as illustrated in Fig. 1.

Unfortunately, this response-time bound is not directly useful since it depends on the relative arrival time A, which is unknown *a priori*. To obtain a general response-time bound, we must determine the value of A for which F is maximized.

To this end, observe that A is an offset *within* $J_{i,j}$'s busy window; the space of possible values of A is thus bounded by the maximum busy-window length. To bound the length of the busy window, consider the least positive constant L such that

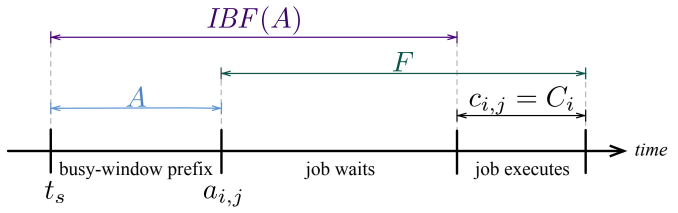


Fig. 1: Worst-case scenario for the job $J_{i,j}$ under analysis. $J_{i,j}$'s busy window starts at time t_s with the arrival of a higher-priority job, $J_{i,j}$ arrives at time $a_{i,j} = t_s + A$, and finishes at time $a_{i,j} + F$ after executing for $c_{i,j} = C_i$ time units. Interfering higher-priority jobs execute for $\mathit{IBF}(A)$ time units in total during $[t_s, t_s + A + F)$.

$$L = \sum_{\tau_k \in \tau} \mathit{RBF}_k(L). \tag{3}$$

If such an L exists, which is the case in non-overloaded systems, then the total workload generated in an interval of length L does not exceed the amount of work the processor can carry out in the interval, which implies instants with zero backlog are spaced at most L time units apart. In a non-overloaded system, a busy window's length is hence bounded by L.

Nonetheless, the resulting search space $0 \le A < L$ is still too large since L can have a very large magnitude in practice (*e.g.*, if parameters are expressed in processor cycles). To further reduce the search space, observe that different values of A can map to the same value of $\mathit{IBF}(A)$ since the underlying arrival curves are step functions. It hence suffices to consider only relative arrival times that map to distinct values of $\mathit{IBF}(A)$. After these two reductions, we obtain a *sparse* search space

$$\mathcal{A} \triangleq \{A < L \mid \exists \tau_k \in \tau,\ \mathit{RBF}_k(A) \neq \mathit{RBF}_k(A + \varepsilon)\} \tag{4}$$

within which it is feasible to perform an exhaustive search. Let R denote the search result, *i.e.*, the least positive value s.th.

$$\forall A \in \mathcal{A},\ \exists F,\ A + F = \sum_{\tau_k \in \tau} \mathit{RBF}_k(A + \varepsilon) \ \wedge\ F \le R. \tag{5}$$

Equivalently, $R = \max_{A \in \mathcal{A}} \{ (\sum_{\tau_k \in \tau} \mathit{RBF}_k(A + \varepsilon)) - A \}$. The general response-time bound for FIFO scheduling with arbitrary deadlines and arbitrary arrival curves follows.

Theorem 1. *If a finite bound L on the maximum busy-window length exists, then any job $J_{i,j}$ of any given task $\tau_i \in \tau$ will finish execution by time $a_{i,j} + R$.*

C. The Need for Verification

FIFO is a fairly simple policy, and as there are no surprises in Theorem 1, the proposed RTA may seem readily *plausible* to an experienced scholar of real-time systems. Nonetheless, it is still necessary to *rigorously* justify its correctness: in the past, there have been far too many cases where intuitive results initially appeared correct, only later to be found subtly flawed [12].

Case in point, both the classic CAN analysis [18, 50] and several results on self-suspending tasks [14]—results that were widely believed to be correct for many years, in some cases even decades—are infamous examples of incorrect RTAs that in terms of technique and style of derivation (*i.e.*, appeal to intuition) do not substantially differ from Theorem 1.

So should Theorem 1 be trusted? Even though FIFO is simple, there is still opportunity for mistakes large and small. Consider Eq. (2): does it really account for all self-interference and possible ties in priority, even if multiple jobs of a bursty task arrive at the same time? Do Eqs. (3) and (4) properly take into account the effects of all *push-through blocking* that arises in the absence of preemptions, which felled the original CAN analysis [18]? Are A and F *offsets* or *interval lengths*, and are the corresponding analysis windows half-open or closed? From own experience, these details can be quite confusing and all too easily cause off-by-one errors—*e.g.*, is it indeed $A + \varepsilon$ in Eq. (5)? Similarly, what if in Eq. (4) the steps of RBF_k were instead given by $RBF_k(A - \varepsilon) \neq RBF_k(A)$? Chances are it would seem just as plausible.

To mitigate the risk of human error, the traditional approach has been to rely on "pen-and-paper" proofs and peer review. However, experience shows that peer reviewers, despite their best efforts, cannot reliably spot all correctness issues [12]. In particular, if the claimed results seem plausible enough, and if the provided "proofs" really are just convincingly sounding, well-flowing appeals to intuition (rather than rigorous, step-by-step mathematical derivations), there is a limit to the level of scrutiny that even the best-intentioned, but time-constrained peer reviewers can achieve. Unsurprisingly, there are examples of flawed analyses passing peer review multiple times both at the conference and the journal levels [28] and finding their way into industrial practice [18].

Since real-time systems are deployed in safety-critical applications, such mistakes are not just expensive or embarrassing, but potentially disastrous. Generally, they undermine the trust in formal timing analysis. Why should a certification authority accept a response-time bound as evidence of a critical task's temporal correctness if the RTA used to obtain the bound is itself clouded by reasonable doubt?

To overcome this unsatisfying state of affairs, we would want an RTA backed by a proof of correctness that is not rooted in intuition and not subject to human error. Fortunately, owing to advances in the field of interactive theorem proving, this vision is quickly becoming reality. In particular, with modern interactive proof assistants and powerful frameworks, it is now possible to create *mechanized* proofs (*i.e.*, formal proofs that can be automatically checked by machines) for practical problems with reasonable effort.

Nonetheless, the barrier to entry remains significant, as working with a proof assistant represents a new way of reasoning that few researchers in the real-time systems community have experience with. To make matters worse, there are few good examples to emulate, as prior work in this area has focused primarily on verified results *per se*, but not on *how* they were verified. We therefore dedicate the remainder of this paper to a case study in RTA verification—the first of its kind—with the aim of making formal verification more accessible.

Ultimately, by using the COQ proof assistant and building on the PROSA project [2, 12] and its aRTA framework [10], we were able to mechanize Theorem 1 and thereby create the first formally verified RTA for FIFO scheduling in less than 400 lines of COQ code (including proofs)—formal, machine-checked analysis is more viable than many realize today. With the following case study, we seek to explain our mechanized derivation and shed light on the involved steps, with the hope that it may be emulated in future work.

III. FOUNDATIONS OF THE VERIFICATION

We begin with a brief introduction to the building blocks underlying our verification effort: the COQ proof assistant, PROSA, and its aRTA framework.

Developing a machine-checked proof with COQ entails **(1)** stating mathematical definitions and theorems, and **(2)** manually constructing proof scripts that establish the stated theorems in a manner that can be automatically verified by COQ. While (1) is done using GALLINA, the native functional programming language of COQ, (2) is usually done by steering the proof engine using LTAC, the language of COQ *tactics*.[1] Although knowledge of LTAC is essential to *constructing* proofs, it is not required to *understanding* the formal development and higher-level arguments at the specification level (*i.e.*, definitions and theorem statements), which is the aim of this paper. Hence, we omit a description of LTAC and refer the interested reader to existing introductions to proving with LTAC [39, 46].

A. An Introduction to GALLINA *as used in* PROSA

COQ's specification language GALLINA is a strongly typed functional programming language based on the *calculus of inductive constructions* [15, 16, 45]. We focus here on the essential elements required to understand the specifications shown in this paper. As a running example, Listing 1 exhibits the main programming constructs used frequently in PROSA.

PROSA follows an axiomatic approach that resembles how real-time scheduling problems are typically defined in the literature. Specifically, this means that the central entities of "jobs" and "tasks" are defined as opaque types about which no implicit semantic assumptions are made. This can be seen in Line 1 of Listing 1, which defines `JobType` as an alias of `eqType`. This, in turn, implies that a typing statement of the form `Job : JobType` (*e.g.*, in Line 7) means that no assumptions are made about `Job` other than that different elements of `Job` (*i.e.*, jobs) can be told apart. In other words, the type `Job` must support *decidable equality*, hence the name `eqType`.

Any parameters or assumptions, such as on WCETs, arrival times, *etc.*, must be stated explicitly. To express that it is assumed that jobs have certain attributes, such as an arrival time, PROSA uses *type classes* [48]. For example, Line 3 defines a type class named `JobArrival`, which on Line 4 introduces a function `job_arrival` that maps each job (of type `Job`) to a point in time (of type `instant`). In PROSA, all parameters are introduced one by one in this axiomatic manner.

[1]For the sake of completeness, we note that there exist alternatives to LTAC such as MTAC [34, 54]. Furthermore, it is in fact also possible to provide low-level proof terms directly in COQ's native logic without the help of any tactic language. However, these advanced alternatives are less relevant to beginning users and currently not used in PROSA, which relies exclusively on LTAC.

```
1  Definition JobType ≔ eqType.
2
3  Class JobArrival (Job : JobType) ≔
4   job_arrival : Job → instant.
5
6  Section BasicLemma.
7   Context {Job : JobType}.
8   Context `{JobArrival Job}.
9
10  Definition arrived_between (j : Job) (t1 t2 : instant)
11   ≔ t1 ≤ job_arrival j < t2.
12  Definition arrived_before (j : Job) (t : instant)
13   ≔ job_arrival j < t.
14
15  Variable t1 t2 : instant.
16
17  Hypothesis H_t1_le_t2 : t1 < t2.
18
19  Lemma arrived_between_before :
20    ∀ j, arrived_between j t1 t2 → arrived_before j t2.
21  Proof. [. . . ] Qed.
22
23 End BasicLemma.
```

Listing 1: Illustrative GALLINA code resembling PROSA.

COQ's type classes are similar to those in HASKELL or interfaces in JAVA, meaning that they stipulate the existence of named functions without giving a concrete definition, which enables generic programming and abstract reasoning (*i.e.*, subsequent proofs must hold for *any* possible implementation). Fortunately, an in-depth understanding of the semantics of type classes [48] is not essential to their use in PROSA. At an intuitive level, PROSA simply uses type classes to explicitly list the job and task parameters that an analysis relies on, just like Sec. II-A explicitly introduces each relevant parameter. We will discuss a concrete example demonstrating this in Sec. III-B.

Next, Lines 6 and 23 define a *section*. In GALLINA, the section mechanism is used to define the scope for *context declarations*, *variables*, and *hypotheses*, which we will discuss shortly. Intuitively, a section is a means to group related definitions and theorems that make common assumptions.

For example, the context declarations in Lines 7 and 8 express that the subsequent code assumes the existence of a set of jobs (the `Job` type, Line 7), where each such job is assumed to have an arrival time (`JobArrival` in Line 8 refers to the type class defined in Line 3). Once a job attribute has been introduced by means of a context declaration, the corresponding function can be used in the definitions and claims enclosed in the section. For example, Lines 10 and 12 define two predicates on a given job's arrival time by referring to the function `job_arrival`. As a result, these predicates are generic — they apply to any set of jobs and any concrete arrival times.

An aside on notation: by convention, the PROSA codebase uses simply j to denote a given job. To stay close to the code, we adopt this scheme when discussing GALLINA listings and in the following use j and $J_{i,j}$ interchangeably to refer to jobs.

Next, we look at two core constructs in GALLINA, variables and hypotheses, which PROSA uses extensively. The variable declaration on Line 15 introduces two variables t_1 and t_2 of type `instant`. PROSA uses a discrete time model and thus time in PROSA is modeled by the natural numbers, so that `instant` is just an alias of `nat`, the type of natural numbers.

COQ's section variables are variables in the mathematical sense, meaning that they do not "store" values (as in a procedural programming language). Rather, statements about variables must hold for any possible value assignment (*i.e.*, section variables are implicitly $\forall$-quantified). The variables t_1 and t_2 thus represent *any* two points in time (or even the same).

To restrict generality — that is, to state assumptions on variables — COQ provides the **`Hypothesis`** keyword. For example, on Line 17, a *hypothesis* named `H_t1_le_t2` relating the two variables t_1 and t_2 is imposed. The hypothesis is given a name (by convention, starting with the prefix "`H_`") so that it can be easily referenced in proofs.

Hypotheses in COQ are propositions. They express preconditions that must be satisfied in order for subsequent proofs to hold. For example, Line 17 stipulates $t_1 < t_2$: the following lemma can thus rely on the assumption that the two instants describe a non-empty interval, with t_1 denoting the start point.

Finally, a (trivial) lemma named `arrived_between_before` is stated on Line 19. In natural language, this lemma observes that any job j that arrives *between* t_1 and t_2 (as defined in Line 10) must necessarily arrive *before* t_2 (Line 12). The type of job j is not given explicitly here because COQ can derive it via type inference. Note that the keywords **`Theorem`**, **`Lemma`**, **`Fact`**, and **`Corollary`** are equivalent in COQ and used interchangeably.

Following the statement of a lemma, a proof of the lemma must be given enclosed by the keywords **`Proof`** and **`Qed`**. The proof itself proceeds with the help of LTAC tactic applications, which we omit here. The command **`Qed`** invokes COQ's proof checker, which verifies the LTAC-generated proof of the lemma. If the proof checker succeeds, the lemma is saved and can then be used in the proofs of subsequent lemmas of theorems.

Variables and hypotheses are *scoped* to their enclosing section, *i.e.*, variables and hypotheses can be used in the definitions of other variables, hypotheses, and lemmas only within the same section. For example, the lemma `arrived_between_before` is stated in terms of the variables t_1 and t_2, and holds only if the two variables satisfy the hypothesis `H_t1_le_t2`.

If a lemma is to be used *outside* its defining section, COQ ensures that all stated assumptions are met, *i.e.*, it is impossible to "forget" about any restrictions of generality under which a lemma was proven. For example, if `arrived_between_before` is to be used in a proof elsewhere, then `H_t1_le_t2` becomes a *proof obligation* that must be shown to hold.

With these few GALLINA basics in place, we are now ready to take a look at the actual PROSA model of real-time systems.

B. The PROSA *Axiomatic Model of Real-Time Scheduling*

Listing 2 summarizes key definitions in PROSA relevant to our work. As already discussed, **jobs** (Line 1) and **tasks** (Line 2) are opaque types in PROSA upon which no implicit semantic assumptions are made (other than decidable equality).

Lines 4–11 introduce the key **job parameters** `job_cost`, `job_arrival`, `job_deadline`, and `job_task` using type classes (as just explained in Sec. III-A). The first three job parameters respectively correspond to $c_{i,j}$, $a_{i,j}$, and $d_{i,j}$ (recall Sec. II-A).

```
 1 Definition JobType ≔ eqType.
 2 Definition TaskType ≔ eqType.
 3
 4 Class JobCost (Job : JobType) ≔
 5  job_cost : Job → work.
 6 Class JobArrival (Job : JobType) ≔
 7  job_arrival : Job → instant.
 8 Class JobDeadline (Job : JobType) ≔
 9  job_deadline : Job → instant.
10 Class JobTask (Job : JobType) (Task : TaskType) ≔
11  job_task : Job → Task.
12
13 Class TaskCost (Task : TaskType) ≔
14  task_cost : Task → duration.
15 Class TaskDeadline (Task : TaskType) ≔
16  task_deadline : Task → duration.
17
18 Definition valid_job_cost (j : Job) ≔
19  job_cost j ≤ task_cost (job_task j).
20
21 Definition arrival_sequence ≔ instant → seq Job.
22 Class MaxArrivals (Task : TaskType) ≔
23  max_arrivals : Task → duration → nat.
24 Definition respects_max_arrivals arr_seq τ_i ≔ ∀t_1 t_2,
25   t_1 ≤ t_2 → number_of_task_arrivals arr_seq τ_i t_1 t_2
26                 ≤ max_arrivals τ_i (t_2 - t_1).
27
28 Section Schedule.
29   Definition processor_state ≔ option Job.
30   Definition schedule ≔ instant → processor_state.
31
32   Variable sched : schedule.
33   Variable j : Job.
34   Definition scheduled_at t ≔ sched t == Some j.
35   Definition service t ≔ ∑(0 ≤ t' < t) scheduled_at t'.
36   Definition completed_by t ≔ service t ≥ job_cost j.
37 End Schedule.
38
39 Definition response_time_bounded_by τ R ≔
40  ∀ sched j, job_task j = τ →
41             completed_by sched j (job_arrival j + R).
```

Listing 2: Key constructs in PROSA. These definitions, here grouped together and streamlined for presentation purposes, are actually distributed across several files in the PROSA code base. The types `work`, `instant`, and `duration` are aliases of natural numbers (`nat`).

The latter one, `job_task`, maps each job to its associated task, and hence conceptually corresponds to the task index "i".

Similarly, Lines 13–16 introduce the **task parameters** `task_cost` and `task_deadline`, which correspond to C_i and D_i, respectively. An essential aspect of the PROSA specification is to encode the *semantics* of all model parameters, which it accomplishes by stating hypotheses that *relate* the various parameters. To this end, PROSA defines **validity constraints**. For example, Line 18 defines the notion of a correct WCET bound: a given job j's cost is deemed *valid* if it does not exceed the WCET of its associated task (*i.e.*, $c_{i,j} \leq C_i$).

There are too many validity constraints to summarize them all here, but it is important to realize that PROSA's axiomatic semantic model is simply a (large) collection of such validity constraints. Together, the validity constraints encode the meaning of the model stated in Sec. II-A. Lemmas in turn use this semantic model by stating the necessary validity constraints as hypotheses in their enclosing sections.

Lines 21–26 formalize the **arrival curve** concept. First, Line 21 defines the notion of an **arrival sequence**, which is simply a function that, given a time, yields the sequence of jobs that arrive at that time. Line 22 defines the task parameter `max_arrivals`, which corresponds to α_i. Finally, Lines 24–26 give the validity constraint: given an arrival sequence `arr_seq` and a task τ_i, `max_arrivals` is a valid arrival curve if the number of jobs of τ_i that arrive in any interval $[t_1, t_2)$ does not exceed `max_arrivals` τ_i $(t_2 - t_1)$, which is equivalent to Eq. (1). (The definition of `number_of_task_arrivals` is omitted here.)

Next, the section in Lines 28–37 introduces the notion of an ideal uniprocessor **schedule**. In PROSA, a schedule (Line 30) is a function that maps each point in time (*i.e.*, a natural number) to a *processor state*. Under the ideal uniprocessor model, a **processor state** (Line 29) is simply the scheduled job (if any), which is expressed with COQ's `option` type, a standard construct in functional programming languages expressing the possible absence of a value. A value of type `option Job` is either the constant `None`, which indicates that no job is scheduled (*i.e.*, the processor is idle), or a value `Some` j, which represents that the job j is running at the time.

The next three definitions apply to any given schedule `sched` and any given job j, as stated in Lines 32–33. The definition on Line 34 simply makes the semantics of the schedule explicit: job j is *scheduled at* time t if `sched t`, the processor state at time t, equals `Some j`. In turn, the function `service` defined on Line 35 measures the amount of service received by job j up to time t. On an ideal uniprocessor, j receives exactly one unit of processor service in each instant that it is executing. The function `service` hence simply counts the number of times $t' < t$ such that j is scheduled at time t'—the syntax `∑(0 ≤ t' < t)` is equivalent to $\sum_{0 \leq t' < t}$. As a result, we obtain a natural definition of **job completion**: as stated in Line 36, a job is complete by time t if the amount of service it has received prior to t is no less than its execution cost.

Finally, Lines 39–41 show PROSA's formalization of the principal subject of this paper: a constant R is a **response-time bound** for a task τ if, for any job j of τ and in any arbitrary schedule `sched`, it holds that job j is complete within at most R units of its arrival, *i.e.*, by time `job_arrival` j + R.

Due to space constraints, some details have been elided in Listing 2. Nonetheless, it provides a faithful summary of the parts of the PROSA semantic model most relevant to this paper.

C. Abstract Response-Time Analysis (ARTA)

To establish that a claimed response-time bound holds, one could prove a corresponding RTA "from scratch" (*i.e.*, starting from first principles). However, this would involve a lot of repetitive and ultimately unnecessary reasoning, and hence is unattractive in practice. Instead, PROSA provides the ARTA framework [10] to streamline the development of new RTAs.

Intuitively, ARTA can be seen as a "template" for RTAS. Technically, it is a general proof of the classic busy-window principle that proceeds at an *abstract* level of reasoning by relying on only a small set of assumptions (*i.e.*, variables and hypotheses). What makes ARTA "abstract" is what it does *not* rely on: a specific scheduling policy or preemption model.

aRTA instead builds on a *behavioral interface* comprised of three abstract functions, namely the *interference*, *interfering workload*, and *interference bound* functions, that jointly characterize the effects of scheduling decisions. To obtain a formally verified RTA for a specific scheduling policy, aRTA can be *instantiated*, which means, firstly, concretely defining these functions to capture the semantics of the targeted policy and, secondly, proving that all analysis assumptions upon which aRTA rests are satisfied by the chosen definitions.

The behavioral interface can capture a wide variety of system models. Indeed, prior work has instantiated aRTA for FP and EDF scheduling in combination with fully-preemptive and fully non-preemptive tasks, segmented limited-preemptive tasks, and floating non-preemptive sections [10]. In this paper, we provide further evidence for the generality of aRTA by introducing the first instantiation of aRTA for FIFO scheduling.

IV. Instantiating aRTA: A Step by Step Breakdown

In this section, we take a close look at how we instantiated aRTA to verify Theorem 1. As our case study is intended to serve as a template for future efforts, we describe not only the steps specific to the FIFO instantiation, but also more generally a process other instantiations may follow. Our description is split across seven subsections, each dedicated to one conceptual step, building up to the final Theorem 2. Throughout this section, we refer to Listing 3, which shows excerpts from our Coq development. The full development is available online.[2]

A. Defining the System Model

Before any formal claims can be stated, an initial setup is needed to define the system model under consideration. As explained in Sec. III-B, the system model is established by introducing section variables and hypotheses expressing validity constraints. For example, Lines 1–3 in Listing 3 introduce the task under consideration τ_i, the task set τ, and the schedule and arrival sequence under analysis. Since section variables are implicitly $\forall$-quantified, this context ensures generality—*e.g.*, the following must hold for *any* semantically valid schedule.

As mentioned previously, the semantics of the system model is encoded by defining validity constraints as hypotheses. For example, the hypothesis `H_work_conserving` on Lines 5–8 states that we assume the schedule under consideration to be work-conserving. It may be read as follows: for any job j and at any point in time t, if j is a job relevant to the schedule (*i.e.*, it stems from the arrival sequence introduced in Line 2), and if j is backlogged (pending but not scheduled) at time t in `sched` (the schedule under consideration, introduced in Line 3), then there exists a job j' that is scheduled at time t. Recall from Sec. II-A that a work-conserving schedule is one of the central assumptions underlying the proposed RTA—hypothesis `H_work_conserving` formalizes this assumption.

[2]Please refer to https://people.mpi-sws.org/~bbb/papers/details/rtss22 for supplemental materials, which include the complete proof, instructions explaining how to check it, and a cross-referenced HTML rendering of the Coq development that is helpful when following along with the discussion in Sec. IV.

The system model further specifies the type of processor under consideration (*e.g.*, ideal uniprocessor), workload model (*e.g.*, sporadic tasks), scheduling policy (*e.g.*, FIFO), preemption model, *etc.* We define each of these elements to match the system model stated in Sec. II-A. For example, recall from Listing 2 the processor model (Line 29 in Listing 2) and the workload model based on arrival curves (Lines 21–26 of Listing 2). Due to space constraints, these variables and hypotheses have been elided in Listing 3.

The scheduling policy and preemption model are obviously key elements of the system model. As mentioned previously, the core aRTA theorem abstracts from these specifics, but an instantiation usually specifies both concretely and must then prove that aRTA's abstract assumptions are satisfied by the concrete definitions (as we will see shortly in Sec. IV-B).

In the case of FIFO, the situation is slightly different. Since the preemption model of a system is irrelevant to the FIFO RTA (a FIFO scheduler does not preempt even if the workload *per se* is preemptable), our aRTA instantiation concretely defines the FIFO scheduling policy, but leaves the preemption model $\forall$-quantified (*i.e.*, abstract). As a result, the verified RTA applies to *any* preemption model and thus maximizes generality.

With this setup in place, the objective of our aRTA instantiation is to formally state and prove a bound on the response time of task τ_i that is equivalent to Theorem 1.

B. Encoding the Scheduling Policy and Preemption Model

The first step is to connect the scheduling policy to aRTA.

aRTA's behavioral interface. Recall from Sec. III-C that aRTA models the scheduling policy *implicitly* via its behavioral interface, which captures the *effects* of scheduling decisions. The interface consists of three functions, which the aRTA core assumes as section variables, together with some hypotheses relating these functions. To apply aRTA, we must provide concretely defined functions and show that our chosen definitions comply with aRTA's hypotheses.

In this subsection, we define the first two required functions: the *interference* $\mathcal{I}_\sigma$ and the *interfering workload* $\mathcal{W}_\sigma$. (The subscript σ signifies that the definitions are made in the context of a given schedule $\sigma =$ `sched`; we retain it here to remain consistent with Bozhko and Brandenburg's notation [10].) Intuitively, $\mathcal{W}_\sigma$ tracks how "delay" potentially affecting the task under analysis is being *produced*, while $\mathcal{I}_\sigma$ tracks how such "delay" is being *consumed* over time.

More formally, consider a job $J_{i,j}$ of the task under analysis τ_i and an arbitrary point in time t. The interference function $\mathcal{I}_\sigma(J_{i,j}, t)$ is a predicate that aRTA requires to be true at time t iff $J_{i,j}$ cannot be scheduled at time t due to *some* unspecified source of delay (irrespective of whether $J_{i,j}$ is actually pending at time t). Correspondingly, the interfering workload function $\mathcal{W}_\sigma(J_{i,j}, t)$ counts the amount of interference that is *generated* at time t w.r.t. $J_{i,j}$ (*e.g.*, the amount of higher-priority workload released at time t). In other words, $\mathcal{W}_\sigma$ counts the number of future points in time at which $\mathcal{I}_\sigma$ is true due to "delay" introduced at time t. These abstract concepts will become much clearer with a concrete example.

```
 1  Variables (τ_i : Task) (τ : seq Task).
 2  Variable arr_seq : arrival_sequence Job.
 3  Variable sched : schedule (ideal.processor_state Job).
 4
 5  Hypothesis H_work_conserving :
 6   ∀ j t, arrives_in arr_seq j →
 7   backlogged sched j t →
 8   ∃ j', scheduled_at sched j' t.
 9
10  Definition is_priority_inversion t :=
11   if sched t is Some j_lp then ¬hep_job j_lp j else false.
12  Definition interf_hep_job j t :=
13   if sched t is Some j_hp then another_hep_job j_hp j
14   else false.
15  Definition interference j t :=
16   is_priority_inversion j t || interf_hep_job j t.
17
18  Definition int_wl_hep_jobs j t :=
19   ∑(j' ← arrivals_at arr_seq t | another_hep_job j' j)
20   job_cost j'.
21  Definition interf_workload j t :=
22   is_priority_inversion j t + int_wl_hep_jobs j t.
23
24  Fact abstractly_work_conserving :
25   work_conserving_ab interference interf_workload.
26
27  Variable L : duration.
28  Hypothesis H_L_positive : L > 0.
29  Hypothesis H_fixed_point : L = total_rbf L.
30  Fact bi_bounded : busy_intervals_bounded_by L.
31
32  Definition IBF A Δ :=
33   (∑(τ_k ← τ) rbf τ_k (A + ε)) - task_cost τ_i.
34  Lemma interference_bounded :
35   job_interference_bounded_by IBF.
36
37  Definition is_in_concrete_search_space A :=
38   (A < L) ∧ has (λ τ_k ⇒ rbf τ_k A ≠ rbf τ_k (A + ε)) τ.
39  Lemma A_is_in_concrete_search_space :
40   ∀ A, is_in_abstract_search_space A →
41          is_in_concrete_search_space A.
42
43  Variable R : duration.
44  Hypothesis H_R_max :
45   ∀ A, is_in_concrete_search_space A →
46          ∃ F, A + F ≥ ∑(τ_k ← τ) rbf τ_k (A + ε) ∧ F ≤ R.
47
48  Lemma soln_abstract_response_time_recurrence :
49   ∀ A, is_in_abstract_search_space A →
50          ∃ F, A + F ≥ task_rtct τ_i + IBF τ_i A (A + F)
51                 ∧ F + (task_cost τ_i - task_rtct τ_i) ≤ R.
52
53  Theorem FIFO_uniprocessor_response_time_bound :
54    response_time_bounded_by τ_i R sched.
```

Listing 3: Streamlined excerpts from the FIFO instantiation of ARTA.[2]

Defining $\mathcal{I}_\sigma$ and $\mathcal{W}_\sigma$. Lines 15–16 in Listing 3 specify $\mathcal{I}_\sigma$ for our FIFO instantiation: a given job j is considered to incur interference iff it experiences priority inversion (Lines 10–11) or a higher-priority job is scheduled (Lines 12–14). Here, the auxiliary predicate `another_hep_job` j_{hp} j in Line 13 is equivalent to $j_{hp} \neq j \wedge$ `job_arrival` $j_{hp} \leq$ `job_arrival` j, and $\neg$`hep_job` j_{lp} j in Line 11 implies `job_arrival` $j_{lp} >$ `job_arrival` j ("$\neg$" denotes boolean negation).

As an aside, it may seem strange that our proof reasons about priority inversion although priority inversion is actually impossible under FIFO scheduling. This could in fact be avoided and is the consequence of a design choice: our proof touches on priority inversion because we reuse general, ARTA-provided definitions for arbitrary *job-level fixed-priority* (JLFP) policies. Since JLFP policies such as EDF and FP scheduling can be subject to priority inversion when jobs execute non-preemptively, the general definitions require us to consider priority inversion explicitly and prove its absence. While conceptually a digression, the reuse of existing JLFP definitions is a worthwhile tradeoff as it saves a significant amount of proof effort. Specifically, it enables the reuse of general results already present in PROSA for some of the following proof obligations.

The interfering workload function $\mathcal{W}_\sigma$ is defined in Lines 21–22 to match the conditions that cause $\mathcal{I}_\sigma$ to be true. Since structurally there are two cases in Line 16, there are also two bounds summed in Line 22. The first summand is simply the priority-inversion predicate itself, which again is later shown to always evaluate to *false*. (There is an implicit type-conversion at work here: the boolean predicate is used as a natural number, with the obvious interpretation of *true* $\mapsto 1$ and *false* $\mapsto 0$.) The second summand is the more interesting `int_wl_hep_jobs`, defined in Lines 18–20. The syntax in Lines 19–20 defines a sum across all jobs j' arriving at time t ("$j' \leftarrow$ `arrivals_at arr_seq` t") that satisfy "`another_hep_job` j' j," adding up their actual execution costs ("`job_cost` j'").

Observe how $\mathcal{I}_\sigma$ and $\mathcal{W}_\sigma$ are designed to match: for example, let j denote any job and suppose that at time $t = 10$ a single higher-priority job j' arrives (w.r.t. j), and that j' requires 3 units of service. Then $\mathcal{W}_\sigma(j, 10)$ evaluates to 3—which is exactly the number of times that $\mathcal{I}_\sigma(j, t')$ can evaluate to *true* because `sched` t' is `Some` j' (recall Line 13), since after this has happened 3 times, j' is necessarily complete. Equivalently, in simpler terms, the job j' arriving at time $t = 10$ can cause job j to incur up to 3 time units of delay.

C. From Classic to Abstract Work Conservation

The relationship sketched above, namely that $\mathcal{I}_\sigma$ and $\mathcal{W}_\sigma$ always eventually match, is a central pillar of ARTA and referred to as "abstract work conservation." We must prove it.

Abstract busy window. In order to state the proof obligation precisely, let $C_{\mathcal{I}}(J_{i,j}, [t_1, t_2)) \triangleq \sum_{t_1 \leq t < t_2} \mathcal{I}_\sigma(J_{i,j}, t)$ and $C_{\mathcal{W}}(J_{i,j}, [t_1, t_2)) \triangleq \sum_{t_1 \leq t < t_2} \mathcal{W}_\sigma(J_{i,j}, t)$ denote *cumulative* interference and workload, respectively.

Now, consider a point in time t where $C_{\mathcal{I}}(J_{i,j}, [0, t)) = C_{\mathcal{W}}(J_{i,j}, [0, t))$ and either $t \leq a_{i,j}$ or $J_{i,j}$ is complete. Such a time t is said to be a *quiet time* w.r.t. the job $J_{i,j}$ since all the higher-or-equal priority interference produced so far (*i.e.*, $C_{\mathcal{W}}(J_{i,j}, [0, t))$) has been fully consumed (*i.e.*, no backlog) and $J_{i,j}$ is not carried-in.

Finally, consider an interval $[t_1, t_2)$ such that t_1 and t_2 are quiet times (w.r.t. $J_{i,j}$), no time instant in (t_1, t_2) is quiet (w.r.t. $J_{i,j}$), and $J_{i,j}$ arrives in $[t_1, t_2)$. This interval is called $J_{i,j}$'s *abstract busy window*. Intuitively, $J_{i,j}$ must begin and complete its execution within $[t_1, t_2)$, if $\mathcal{I}_\sigma$ and $\mathcal{W}_\sigma$ are defined properly.

Proof obligation. Formally, the guarantee that $J_{i,j}$ will finish by the end of its busy window is a consequence of *abstract work conservation*, which the instantiation establishes with the fact `abstractly_work_conserving` in Lines 24–25. As mentioned above, we reuse general JLFP definitions for $\mathcal{I}_\sigma$ and $\mathcal{W}_\sigma$. One benefit of this design choice is that the fact `abstractly_work_conserving` is actually a trivial consequence of a general lemma already present in PROSA. For illustrative purposes, let us nonetheless consider the argument in more detail. After unfolding `work_conserving_ab` (*i.e.*, substituting it with its definition), the actual proof obligation is equivalent to:

Lemma 1 (abstract work conservation)**.**

$\forall\, j\ t_1\ t_2\ t,$ `arrives_in arr_seq` $j \rightarrow$
  `job_task` $j = \tau_i \rightarrow$ `job_cost j > 0` $\rightarrow$
  `busy_interval` $j\ t_1\ t_2 \rightarrow\ t_1 \leq$ `t` $< t_2 \rightarrow$
    $\neg$ `interference` $j\ t \leftrightarrow$ `job_scheduled_at` $j\ t$.

In other words, the instantiation must establish that, for any job j of the task under analysis τ_i that arrives and has nonzero execution cost, at any point in time t in j's (abstract) busy window $[t_1, t_2)$, the interference predicate $\mathcal{I}_\sigma$ evaluates to *false* iff job j is scheduled. This lemma ensures on the one hand that $\mathcal{I}_\sigma$ and $\mathcal{W}_\sigma$ indeed "match" since `busy_interval` is phrased in terms of $\mathcal{W}_\sigma$. On the other hand, it also ensures that the definition of $\mathcal{I}_\sigma$ captures the semantics of the scheduling policy since the auxiliary predicate `job_scheduled_at` (with obvious meaning, omitted from Listing 3) depends on the policy's rules. As one may expect, the proof of Lemma 1 proceeds from classic work conservation (`H_work_conserving`, Lines 5–8).

D. Bounding the Maximum Busy-Window Length

The next step is conceptually straightforward. As already discussed in Sec. II, RTA is applicable only if busy windows are finite. To this end, aRTA simply assumes the existence of some bound L on the maximum length of any abstract busy window. An instantiation in turn must define an actual bound, which obviously depends on the concrete workload properties and the analyzed scheduling policy, and prove it correct.

Listing 3 shows the relevant parts in Lines 27–30. Mirroring the setup before Eq. (3) in Sec. II, Line 27 first introduces a section variable L, which is constrained to be positive (Line 28). Crucially, `H_fixed_point` on Line 29 then formalizes the assumption that Eq. (3) holds. The fact `bi_bounded` on Line 30 then establishes the correctness of the bound. After unfolding and slight simplification, it is equivalent to:

Lemma 2 (max. length of abstract busy windows)**.**

$\forall\, j,$ `arrives_in arr_seq` $j \rightarrow$
  `job_task` $j = \tau_i \rightarrow$ `job_cost` $j > 0 \rightarrow$
    $\exists\, t_1\ t_2,\ t_2 \leq t_1 + L \wedge$ `busy_interval` $j\ t_1\ t_2$.

That is, the instantiation must prove that, for any job j of the task under analysis that arrives and has nonzero execution cost, there exist two points in time t_1, t_2 such that t_2 is within L time units of t_1 and $[t_1, t_2)$ is job j's busy window. Conceptually, the proof proceeds by observing that, if t_1 is the last quiet time before j's release, then another quiet time t_2 necessarily occurs "soon enough" because, jointly, `H_fixed_point`, the definition of $\mathcal{W}_\sigma$, and Lemma 1 imply both that job j completes and that $C_\mathcal{I}(j, [0, t))$ "catches up" with $C_\mathcal{W}(j, [0, t))$.

Technically, `bi_bounded` is again a direct consequence of a general result already available in PROSA, which highlights the benefits of building on top of aRTA's general JLFP definitions.

E. Bounding the Delay Within a Busy Window

Having established that busy windows are finite, the next step is to bound the delay experienced by the job under analysis within its busy window, analogous to Eq. (2). To this end, the third element of aRTA's behavioral interface is the (abstract) *interference-bound function* (*IBF*). As the name suggests, this function must bound abstract interference.

More precisely, let $[t_1, t_2)$ denote the busy window of a job $J_{i,j}$ of the task under analysis, and let $A \triangleq a_{i,j} - t_1$ denote its relative arrival time. aRTA requires a function $\mathit{IBF} : \mathbb{T} \times \mathbb{T} \rightarrow \mathbb{T}$ that bounds cumulative interference: $\forall \Delta$, if $J_{i,j}$ is incomplete at time $t_1 + \Delta$, then $C_\mathcal{I}(J_{i,j}, [t_1, t_1 + \Delta)) \leq \mathit{IBF}(A, \Delta)$.

Since aRTA expects such a function, the FIFO instantiation must provide a concrete definition and prove its correctness (Lines 32–35 of Listing 3). First, Lines 32–33 concretely define *IBF* exactly as stated in Eq. (2): the syntax $\sum(\tau_k \leftarrow \tau)$ is equivalent to $\sum_{\tau_k \in \tau}$, and RBF_k corresponds to `rbf` τ_k. Note that aRTA generally allows *IBF* to depend on Δ, but in the case of the FIFO policy, the concrete definition actually depends only on the relative arrival time A, so that Δ remains unused in Line 33. Correctness of the bound is established in Lines 34–35, which unfolds to:

Lemma 3 (abstract interference bound)**.**

$\forall\, t_1\ t_2\ \Delta\ j,$ `arrives_in arr_seq` $j \rightarrow$ `job_task` $j = \tau_i \rightarrow$
  `busy_interval` $j\ t_1\ t_2 \rightarrow\ t_1 + \Delta < t_2 \rightarrow$
  $\neg$ `job_completed_by` $j\ (t_1 + \Delta) \rightarrow$
    `cumul_interference` $j\ t_1\ (t_1 + \Delta) \leq$ `IBF` $A\ \Delta$.

Here, `cumul_interference` $j\ t_1\ (t_1 + \Delta)$ is $C_\mathcal{I}(J_{i,j}, [t_1, t_1 + \Delta))$. Lemma 3 hence satisfies aRTA's above-stated assumption that *IBF* bounds cumulative interference within busy windows.

F. Defining the Search Space, Abstractly and Concretely

As explained in Sec. II, for the RTA to be *practical*, it is important to obtain a sparse search space. Fortunately, the notion of a sparse search is built into aRTA: this is the *abstract search space*. As in the prior subsections, it's an instantiation's obligation to complement the abstract search space with a *concrete search space* and to relate the two.

To this end, our FIFO instantiation defines a predicate `is_in_concrete_search_space` in Lines 37–38 of Listing 3. Given an offset A, two conditions are checked. First, for A to be included, $A < L$ must hold, since A denotes the relative arrival time of the job under analysis within its busy window, which by Lemma 2 is at most L time units long.

The second condition reads as follows. Given a predicate p, the function `has` $p\ \tau$ checks whether any of the tasks in τ *has* property p, that is, `has` $p\ \tau$ evaluates to *true* iff a task in τ satisfies p. Here, the predicate is $(\lambda\, \tau_k \Rightarrow$ `rbf` $\tau_k\ A \neq$ `rbf` $\tau_k\ (A + \varepsilon))$, which is an anonymous function (*i.e.*, a lambda term) that yields true iff, for a given task τ_k, $RBF_k(A) \neq RBF_k(A + \varepsilon)$. Lines 37–38 hence implement exactly the membership test for the search space given in Eq. (4).

Next, Lines 39–41 relate the concrete to the abstract search space by establishing a subset relationship. Specifically, the lemma shows that any offset A that satisfies the predicate `is_in_abstract_search_space` also satisfies the concrete one given in Lines 37–38. After unfolding the abstract search space's membership predicate, the lemma reduces to:

Lemma 4 (search space inclusion)**.**
$\forall\, A,\ (A = 0 \lor 0 < A < L$
$\quad\land \exists \Delta,$ `IBF` $(A - \varepsilon)\ \Delta \neq$ `IBF` $A\ \Delta \land \Delta < L)$
$\quad\to$ `is_in_concrete_search_space` A.

Lemma 4 thus ensures that the concrete search space correctly captures where the instantiation-provided IBF "steps."

G. Stating and Re-Stating the Claimed Bound

Having established all necessary preliminaries, it is finally time to state the claimed response-time bound R. In a pattern that should be familiar by now, we will characterize R twice: first concretely as we seek to verify it (*i.e.*, matching Sec. II), and then again in the general "shape" that aRTA expects.

To begin with, the concrete definition is given in Lines 43–46. First, R is introduced as a section variable in Line 43. Second, the hypothesis `H_R_max` in Lines 44–46 formalizes the assumption that the claimed bound R satisfies the condition stated in Eq. (5): namely, that for each A in the concrete search space, there exists an offset-specific solution F such that $R \geq F \geq \left(\sum_{\tau_k \in \tau} RBF_k(A + \varepsilon)\right) - A$.

Recall that the overall objective is to show that a value R satisfying `H_R_max` is indeed a response-time bound for the task under analysis. To this end, we seek to apply the general aRTA theorem [10, Theorem 18]. That is, we intend to show that—in the context of all the definitions established in the preceding sections—the claimed response-time bound is a consequence of aRTA's general bound. However, before we can apply the general bound, we must transform the FIFO-specific hypothesis `H_R_max` into the general shape that aRTA expects.

In preparation of this final step, we need to introduce one more concept: the *run-to-completion threshold* $RTCT_i$, which aRTA uses to abstract over many possible preemption models. In short, $RTCT_i$ denotes an upper bound on the amount of processor service that any job of task τ_i must receive before it certainly becomes non-preemptable, after which it runs to completion [10]. As already mentioned in Secs. II-A and IV-A, this notion is actually irrelevant in our case since, under FIFO scheduling, jobs anyway run to completion once started.

Nonetheless, since aRTA is phrased in terms of $RTCT_i$, we must mention it, too. Our FIFO instantiation hence introduces the task parameter `task_rtct` (analogously to Lines 13–16 in Listing 2), but without imposing restrictions on it (beyond basic validity), thereby retaining support for any preemption model.

With the run-to-completion threshold in place, we can restate `H_R_max` in the expected form, which is done in Lines 48–51. Not coincidentally, the structure is similar to `H_R_max`, with three main differences: the statement is about offsets in the abstract search space (rather than the concrete one), the general form uses the abstract `IBF` instead of the FIFO-specific `rbf`, and `task_rtct` is added in Line 50 and subtracted in Line 51. In the case of FIFO, however, `IBF` is defined in terms of `rbf`, and does not use the Δ parameter (recall Sec. IV-E), which greatly simplifies the structure of the inequality. Consequently, with some algebraic manipulation after applying Lemma 4, the proof is able to derive an F that satisfies the stated conditions.

H. Soundness of the Response-Time Bound

Finally, Lines 53–54 of Listing 3 establish the soundness of the claimed response-time bound R using the general aRTA theorem [10, Theorem 18]. In fact, it takes just thirteen lines of Ltac tactic invocations to prove Lines 53–54, which shows the benefit of building on aRTA (rather than proving the RTA from first principles). After unfolding (recall Lines 39–41 in Listing 2), the final theorem of our instantiation translates to:

Theorem 2 (soundness of FIFO RTA)**.**
$\forall\, j,$ `job_task` $j = \tau_i \to$
$\quad$ `completed_by sched` j (`job_arrival` $j + R$).

In other words, any value R that satisfies the hypothesis `H_R_max` (Lines 44–46) bounds the response time of any job of the task under analysis τ_i in the schedule under analysis `sched`. Since we have not placed any restrictions on τ_i or `sched` (besides semantic validity constraints), the bound holds without loss of generality. Furthermore, as the declared system model is equivalent to that assumed in Sec. II-A, and since `H_R_max` matches Eq. (5), Theorem 2 verifies Theorem 1.

V. Discussion

In addition to verifying Theorem 1, our case study also raises some salient points of a more general nature.

Proof effort. Our foremost observation is just how little code was ultimately required to establish the desired result. The module providing the formalized RTA amounts to only 434 non-blank lines, of which 208 are actually comments.[2] The remaining 224 lines include both the Gallina specification discussed herein as well as the Ltac proof scripts actually establishing the lemmas and theorems. Accounting for FIFO-specific definitions and lemmas in Prosa's support libraries adds only another 155 lines of Gallina and Ltac code.

It is surprising, or at least not widely appreciated, that with less than 400 lines of code it is possible to verify an (admittedly simple) RTA. Anecdotally, many researchers still deem the barrier to entry considerably higher. To the contrary, our case study demonstrates that, nowadays, verification of real-time scheduling theory is possible with acceptable effort (once familiarity with Ltac has been obtained, for which high-quality tutorials and textbooks are readily available—see [1, 39, 46]).

Unsurprisingly, the "trick" that allows establishing Theorem 2 with so few lines of code is *maximal code reuse*: by leveraging generic definitions and proofs in Prosa, we could focus on FIFO-specific reasoning. We draw three general conclusions from this. First, it is generally a good idea to start from domain-specific libraries such as Prosa rather than proving everything "from scratch." Second, results that stray further in scope from the existing general results in Prosa will require a proportionally larger proof effort. And third, there is much value in future work extending PROSA's abstract core.

Alternate proof strategy. As mentioned in Sec. I, FIFO arbitration is well-studied in the Network Calculus (NC) literature (*e.g.*, [8]). In fact, Theorem 1 could alternatively also be obtained from NC. However, with the available state-of-the-art COQ libraries, this would not make its verification any easier. Case in point, Roux et al. [47] just recently explored the complexities involved in formalizing NC and formally relating it to RTA. That said, future work extending Roux et al.'s bridging efforts might render NC a viable, and maybe even convenient, starting point for verified RTAs such as Theorem 2.

Limitations and Extensions. For didactic reasons, it is helpful for Theorems 1 and 2 to be as simple as possible. From a practical point of view, however, several extensions are desirable. In particular, we plan to explicitly incorporate scheduling overheads such as context-switching. Additionally, it would be interesting to allow for arbitrary service descriptions based on supply curves (as in NC), which would extend support to reservation-based scheduling [11]. It would be even more ambitious to allow for self-suspensions [14]. In keeping with the spirit of maximal reuse, each of these extensions will primarily require a generalization of PROSA's aRTA core, after which the FIFO-specific reasoning will be relatively simple.

VI. EVALUATION

To assess the proposed RTA empirically, we conducted schedulability experiments with synthetic workloads modeled after the Bosch automotive benchmark due to Kramer et al. [35].

Setup. For a given number of tasks n, we randomly generated each task τ_i as follows. First, we selected a minimum inter-arrival time (or, interchangeably, period) T_i from the set $\{1, 2, 5, 10, 20, 50, 100, 200, 1000\}$ ms according to a (non-uniform) distribution specified by Kramer et al. [35]. The corresponding arrival-bound function is simply $\alpha_i(\Delta) \triangleq \left\lceil \frac{\Delta}{T_i} \right\rceil$.

For each period, Kramer et al. specify a range of periodic-specific average execution times and multiplicative factors to obtain a representative maximum execution time of "runnables," a common sub-task abstraction in automotive systems. For simplicity, we considered each task to consist of one runnable and, given T_i, used Kramer et al.'s tables to randomly generate the task's WCET C_i. The task's resulting utilization is $u_i \triangleq \frac{C_i}{T_i}$. All task parameters were expressed as integral nanoseconds.

For each n, we generated 500 task sets. To each task set and each task, we applied four RTAs: **(i)** the proposed RTA (Theorem 1), **(ii)** Altmeyer et al.'s RTA for sporadic tasks under FIFO scheduling [4], **(iii)** Bozhko and Brandenburg's RTA for non-preemptive FP scheduling (NP-FP) [10], and also **(iv)** their RTA for non-preemptive EDF scheduling (NP-EDF) [10]. We further **(v)** assessed whether finite response-time bounds exist at all by checking the necessary condition $\sum_i u_i \leq 1$.

Baseline comparison. In the first experiment, we varied n from 2 to 30 and compared (i), (ii), and (v), *i.e.*, the two RTAs for FIFO and the feasibility test. The results are shown in Fig. 2a. For (i) and (ii), the graph shows the fraction of task sets for which the RTA could find a response-time bound; for (v), it shows the fraction of feasible task sets.

It is immediately apparent that Altmeyer et al.'s RTA succeeds for far fewer task sets than Theorem 1. In fact, Theorem 1 yields bounds for all feasible workloads (its curve overlaps with the feasibility curve). The underlying reason is that Altmeyer et al.'s RTA does not apply to workloads that can experience self-interference—it can be seen as roughly equivalent to Theorem 1 artificially restricted to a singleton search space $\mathcal{A} = \{0\}$. However, as the number of tasks n increases, so does the total utilization, and consequently self-interference becomes an increasingly frequent phenomenon that renders Altmeyer et al.'s RTA inapplicable. In contrast, by considering the full search, our proposed RTA finds a response-time bound for every workload for which one exists.

To confirm the impact of self-interference, we conducted a second experiment in which we constrained task periods to be at least 20 ms (by discarding and redrawing tasks with $T_i < 20$ ms), which has the effect of rendering workloads much more homogeneous. Due to the period-specific distributions reported by Kramer et al. [35], it further has the effect of significantly lowering the average utilization, which allowed us to vary n from 2 to 80 (in steps of 2).

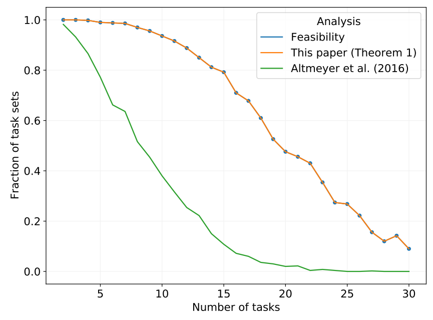
The results are depicted in Fig. 2b. As expected, Altmeyer et al.'s RTA performs much better in this simplified setting, yielding bounds for all task sets with $n \leq 22$. In contrast, in Fig. 2a, it fails to find bounds for more than 60% of the task sets with $n = 10$. Nonetheless, Theorem 1 is still preferable for larger n, yielding bounds for all feasible workloads.

In a third experiment, we kept the restriction to tasks with $T_i \geq 20$ ms, but additionally introduced *release jitter* to reflect workloads with more irregular arrival patterns. Specifically, for each task τ_i, we randomly chose a release jitter $JIT_i \in [0, 0.75T_i]$. The resulting arrival-bound function is $\alpha_i(\Delta) \triangleq \left\lceil \frac{\Delta + JIT_i}{T_i} \right\rceil$. We varied n from 2 to 40; Fig. 2c shows the results.

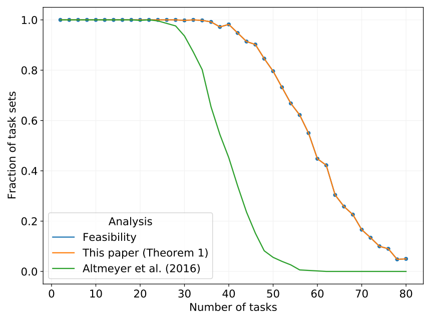
Compared to Fig. 2b, it is striking how the efficacy of Altmeyer et al.'s RTA is dramatically reduced in Fig. 2c: it could find bounds for less than 10% of the task sets with $n = 20$. In contrast, Theorem 1 is completely unaffected by the introduction of release jitter. The explanation is that, on the one hand, release jitter does not affect utilization (*i.e.*, feasibility), while on the other hand, it makes self-interference much more likely. This again demonstrates the advantage of considering the full search space. Overall, we find Theorem 1 to be a welcome improvement over the baseline [4].

Policy comparison. Since Theorem 1 applies to all feasible workloads, it allows (for the first time) to fairly compare FIFO with the more traditional real-time scheduling policies NP-FP (with rate-monotonic priorities) and NP-EDF (with implicit deadlines), as represented by RTAs (iii) and (iv), respectively.

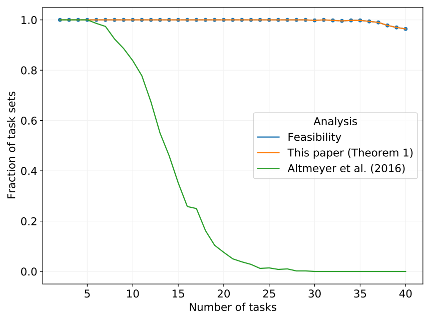
We compared the policies in terms of their average *response-time ratios* (RTRs), which we define as follows. Given a task $\tau_i \in \tau$, let R^{FIFO}, R^{FP}, and R^{EDF} denote the bounds given by RTAs (i), (ii), and (iii), respectively. Task τ_i's RTR under NP-FP (resp., NP-EDF) is then given by $\frac{R^{FP}}{R^{FIFO}}$ (resp, $\frac{R^{EDF}}{R^{FIFO}}$).



(a) Full period range

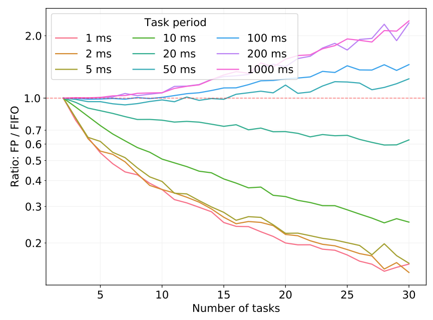


(b) Restricted period range ($T_i \geq 20\,\mathrm{ms}$)

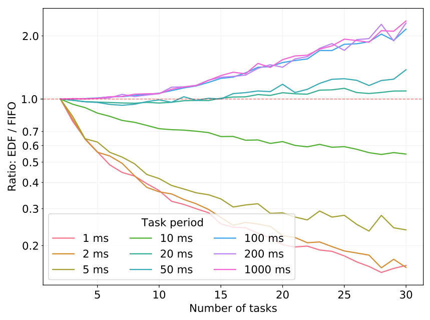


(c) With release jitter $JIT_i \in [0, 0.75T_i]$

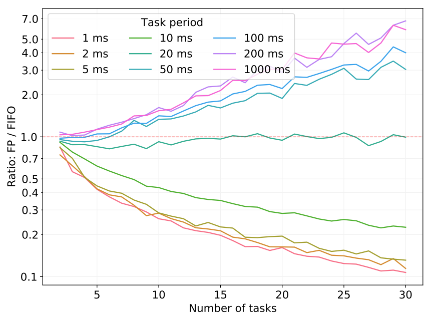
Fig. 2: Comparison of the proposed RTA (Theorem 1) with Altmeyer et al.'s RTA for sporadic tasks under FIFO scheduling [4].



(a) NP-FP



(b) NP-EDF



(c) NP-FP, release jitter $JIT_i \in [0, 1.5T_i]$

Fig. 3: Comparison of FIFO with NP-FP and NP-EDF in terms of average RTR, split by task period.

We tracked the RTRs on a per-period basis and computed the average across all 500 task sets (for each considered n, excluding infeasible task sets).

First, consider Fig. 3a, which shows the average RTR (per period) for NP-FP for the same workloads as considered in Fig. 2a. A clear pattern emerges: the curves are arranged top-to-bottom roughly in order of their periods. The tasks with shorter periods (*i.e.*, higher rates) exhibit lower average RTRs, which means they benefit the most from NP-FP scheduling (instead of FIFO). Conversely, tasks with longer periods (*i.e.*, lower rates) benefit from FIFO scheduling—they exhibit average RTRs exceeding one, which means Theorem 1 yields *lower* bounds than the NP-FP analysis. For the considered workloads, the dividing line is between 20 ms and 50 ms, with the former period benefiting from NP-FP and the latter from FIFO.

Second, consider Fig. 3b, which shows the average per-period RTRs under NP-EDF for the same workloads as considered in Figs. 2a and 3a. The trends are largely the same as in Fig. 3a, with the biggest difference being a less pronounced, but still clearly beneficial average RTR for $T_i = 10\,\mathrm{ms}$. This confirms that NP-EDF and NP-FP are quite alike (for these workloads), but also very different from FIFO.

Next, Fig. 3c shows NP-FP RTRs for a similar setup as in Figs. 2a, 3a and 3b with the addition of release jitter in the range $[0, 1.5T_i]$. Again, release jitter generally increases self-interference and thus worst-case bounds. As a result, the trends remain structurally similar to Fig. 3a, but their magnitude is greatly increased (note the difference in Y-axis scale).

Finally, we plotted all experimental results also as a function of the total utilization $\sum_i u_i$ (rather than n) and observed equivalent trends; due to space constraints we omit these plots.

VII. Conclusion

We have presented a case study in verified analysis of real-time systems, by example of a novel RTA for sporadic tasks under FIFO scheduling. The RTA was derived twice: first in a traditional style based on an appeal to intuition (Sec. II), and once more formally based on Prosa's aRTA framework [10, 12] using the Coq proof assistant (Sec. IV). The case study highlights the similarity of the formal argument to the intuitive line of reasoning. It is our hope that this case study will enable and motivate more researchers to explore the use of proof assistants to minimize human error in mathematical reasoning.

Furthermore, the verified RTA is also of practical interest. Whereas Altmeyer et al.'s RTA for sporadic tasks under FIFO scheduling [4] does not account for self-interference, which limits its applicability (Sec. VI), our new RTA yields a response-time bound as long as the maximum busy window is bounded (*i.e.*, if the workload is feasible at all). Additionally, our experiments show that FIFO scheduling can actually be beneficial for lower-rate tasks, at the expense of higher-rate tasks. With the proposed RTA, workloads that can tolerate this trade-off can benefit from FIFO's low runtime overheads and trivial implementation requirements, which offers engineers a new *trustworthy* alternative for resource-constrained systems.

Acknowledgements

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 803111), and from the German Research Foundation (*Deutsche Forschungsgemeinschaft*, DFG) — 391919384.

References

[1] "The Coq proof assistant," https://coq.inria.fr.

[2] "Prosa," http://prosa.mpi-sws.org/.

[3] T. A. AlEnawy and H. Aydin, "On energy-constrained real-time scheduling," in *Proceedings of the 16th Euromicro Conference on Real-Time Systems (ECRTS)*, 2004, pp. 165–174.

[4] S. Altmeyer, S. M. Sundharam, and N. Navet, "The case for FIFO real-time scheduling," University of Luxembourg, Tech. Rep., 2016, http://hdl.handle.net/10993/24935.

[5] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. J. Wellings, "Applying new scheduling theory to static priority pre-emptive scheduling," *Software engineering journal*, vol. 8, no. 5, pp. 284–292, 1993.

[6] U. N. Bhat, *An introduction to queueing theory: modeling and analysis in applications*. Springer, 2008, vol. 36.

[7] D. P. Bovet and M. Cesati, *Understanding the Linux Kernel: from I/O ports to process management*. O'Reilly, 2005.

[8] M. Boyer, E. Le Corronc, and A. Bouillard, *Deterministic Network Calculus: From Theory to Practical Implementation*. John Wiley & Sons, 2018.

[9] M. Boyer, P. Roux, H. Daigmorte, and D. Puechmaille, "A residual service curve of rate-latency server used by sporadic flows computable in quadratic time for network calculus," in *Proceedings of the 33rd Euromicro Conference on Real-Time Systems (ECRTS)*, 2021, pp. 14:1–14:21.

[10] S. Bozhko and B. B. Brandenburg, "Abstract response-time analysis: A formal foundation for the busy-window principle," in *Proceedings of the 32nd Euromicro Conference on Real-Time Systems (ECRTS)*, 2020, pp. 22:1–22:24.

[11] G. C. Buttazzo, *Hard real-time computing systems: predictable scheduling algorithms and applications*. Springer, 2011.

[12] F. Cerqueira, F. Stutz, and B. B. Brandenburg, "PROSA: A case for readable mechanized schedulability analysis," in *Proceedings of the 28th Euromicro Conference on Real-Time Systems (ECRTS)*, 2016, pp. 273–284.

[13] F. Cerqueira, G. Nelissen, and B. B. Brandenburg, "On strong and weak sustainability, with an application to self-suspending real-time tasks," in *Proceedings of the 30th Euromicro Conference on Real-Time Systems (ECRTS)*, 2018, pp. 26:1–26:21.

[14] J.-J. Chen, G. Nelissen, W.-H. Huang, M. Yang, B. Brandenburg, K. Bletsas, C. Liu, P. Richard, F. Ridouard, N. Audsley *et al.*, "Many suspensions, many problems: a review of self-suspending tasks in real-time systems," *Real-Time Systems*, vol. 55, no. 1, pp. 144–207, 2019.

[15] T. Coquand and G. Huet, "The calculus of constructions," *Information and Computation*, vol. 76, no. 2, pp. 95–120, 1988.

[16] T. Coquand and C. Paulin, "Inductively defined types," in *Proceedings of the International Conference on Computer Logic*, 1988, pp. 50–66.

[17] D. Cornhilll, L. Sha, and J. P. Lehoczky, "Limitations of Ada for real-time scheduling," *ACM SIGAda Ada Letters*, vol. 7, no. 6, pp. 33–39, 1987.

[18] R. I. Davis, A. Burns, R. J. Bril, and J. J. Lukkien, "Controller area network (CAN) schedulability analysis: Refuted, revisited and revised," *Real-Time Systems*, vol. 35, no. 3, pp. 239–272, 2007.

[19] R. I. Davis, S. Kollmann, V. Pollex, and F. Slomka, "Controller area network (CAN) schedulability analysis with FIFO queues," in *Proceedings of the 23rd Euromicro Conference on Real-Time Systems (ECRTS)*, 2011, pp. 45–56.

[20] J. Diemer, D. Thiele, and R. Ernst, "Formal worst-case timing analysis of Ethernet topologies with strict-priority and AVB switching," in *Proceedings of the 7th IEEE International Symposium on Industrial Embedded Systems (SIES)*, 2012, pp. 1–10.

[21] B. Dutertre, "The priority ceiling protocol: formalization and analysis using PVS," in *Proceedings of the 21st IEEE Real-Time Systems Symposium (RTSS)*, 1999, pp. 151–160.

[22] X. Fan, M. Jonsson, and J. Jonsson, "Guaranteed real-time communication in packet-switched networks with FCFS queuing," *Computer networks*, vol. 53, no. 3, pp. 400–417, 2009.

[23] P. Fradet, X. Guo, J.-F. Monin, and S. Quinton, "A generalized digraph model for expressing dependencies," in *Proceedings of the 26th International Conference on Real-Time Networks and Systems (RTNS)*, 2018, pp. 72–82.

[24] P. Fradet, M. Lesourd, J.-F. Monin, and S. Quinton, "A generic Coq proof of typical worst-case analysis," in *Proceedigns of the 39th IEEE Real-Time Systems Symposium (RTSS)*, 2018, pp. 218–229.

[25] P. Fradet, X. Guo, J.-F. Monin, and S. Quinton, "CertiCAN: A tool for the Coq certification of CAN analysis results," in *Proceedings of the 25th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2019, pp. 182–191.

[26] L. George and P. Minet, "A FIFO worst case analysis for a hard real-time distributed problem with consistency constraints," in *Proceedings of 17th International Conference on Distributed Computing Systems (ICDCS)*, 1997, pp. 441–448.

[27] G. Giambene, *Queuing Theory and Telecommunications: Networks and Applications*. Springer, 2005.

[28] A. Gujarati, F. Cerqueira, B. B. Brandenburg, and G. Nelissen, "Correspondence article: a correction of the reduction-based schedulability analysis for APA scheduling," *Real-Time Systems*, vol. 55, no. 1, pp. 136–143, 2019.

[29] M. Günzel and J.-J. Chen, "A note on slack enforcement mechanisms for self-suspending tasks," *Real-Time Systems*, vol. 57, no. 4, pp. 387–396, 2021.

[30] P. K. Harter Jr, "Response times in level-structured systems," *ACM Transactions on Computer Systems (TOCS)*, vol. 5, no. 3, pp. 232–248, 1987.

[31] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst, "System level performance analysis–the SymTA/S approach," *IEE Proceedings-Computers and Digital Techniques*, vol. 152, no. 2, pp. 148–166, 2005.

[32] L. S. Indrusiak, A. Burns, and B. Nikolic, "Analysis of buffering effects on hard real-time priority-preemptive wormhole networks," *arXiv preprint arXiv:1606.02942*, 2016.

[33] M. Joseph and P. Pandya, "Finding response times in a real-time system," *The Computer Journal*, vol. 29, no. 5, pp. 390–395, 1986.

[34] J.-O. Kaiser, B. Ziliani, R. Krebbers, Y. Régis-Gianas, and D. Dreyer, "Mtac2: typed tactics for backward reasoning in Coq," *Proceedings of the ACM on Programming Languages*, vol. 2, no. ICFP, pp. 78:1–78:31, 2018.

[35] S. Kramer, D. Ziegenbein, and A. Hamann, "Real world automotive benchmarks for free," in *Proceedings of the 6th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, 2015.

[36] J. P. Lehoczky, "Fixed priority scheduling of periodic task sets with arbitrary deadlines," in *Proceedings of the 11th IEEE Real-Time Systems Symposium (RTSS)*, 1990, pp. 201–209.

[37] ——, "Scheduling communication networks carrying real-time traffic," in *Proceedings of the 19th IEEE Real-Time Systems Symposium (RTSS)*, 1998, pp. 470–479.

[38] H. Leontyev and J. H. Anderson, "Tardiness bounds for FIFO

scheduling on multiprocessors," in *Proceedings of the 19th Euromicro Conference on Real-Time Systems (ECRTS)*, 2007, pp. 71–71.

[39] A. Mahboubi and E. Tassi, *Mathematical Components*. Zenodo, 2022. [Online]. Available: https://doi.org/10.5281/zenodo.7118596

[40] M. Maida, S. Bozhko, and B. B. Brandenburg, "Foundational response-time analysis as explainable evidence of timeliness," in *Proceedings of the 34th Euromicro Conference on Real-Time Systems (ECRTS)*, 2022, pp. 19:1–19:25.

[41] T. Meisling, "Discrete-time queuing theory," *Operations Research*, vol. 6, no. 1, pp. 96–105, 1958.

[42] M. Nasri, R. I. Davis, and B. B. Brandenburg, "FIFO with offsets: High schedulability with low overheads," in *Proceedings of the 24th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2018, pp. 271–282.

[43] N. Navet and L. Fejoz, "CPAL: High-level abstractions for safe embedded systems," in *Proceedings of the International Workshop on Domain-Specific Modeling*, 2016, pp. 35–41.

[44] N. Navet, L. Fejoz, L. Havet, and A. Sebastian, "Lean model-driven development through model-interpretation: the CPAL design flow," in *Proceedings of the 8th European Congress on Embedded Real Time Software and Systems (ERTS)*, 2016.

[45] C. Paulin-Mohring, "Introduction to the calculus of inductive constructions," in *All about Proofs, Proofs for All*, B. W. Paleo and D. Delahaye, Eds. College Publications, 2015.

[46] B. C. Pierce, A. Azevedo de Amorim, C. Casinghino, M. Gaboardi, M. Greenberg, C. Hriţcu, V. Sjöberg, and B. Yorgey, *Logical Foundations*, ser. Software Foundations, B. C. Pierce, Ed. Electronic textbook, 2022, vol. 1, version 6.2. [Online]. Available: http://softwarefoundations.cis.upenn.edu

[47] P. Roux, S. Quinton, and M. Boyer, "A formal link between response time analysis and network calculus," in *Proceedings of the 34th Euromicro Conference on Real-Time Systems (ECRTS)*, 2022, pp. 5:1–5:22.

[48] M. Sozeau and N. Oury, "First-class type classes," in *International Conference on Theorem Proving in Higher Order Logics*. Springer, 2008, pp. 278–293.

[49] D. Thiele, P. Axer, and R. Ernst, "Improving formal timing analysis of switched Ethernet by exploiting FIFO scheduling," in *Proceedings of the 52nd Annual Design Automation Conference (DAC)*, 2015, pp. 1–6.

[50] K. Tindell, A. Burns, and A. J. Wellings, "Calculating controller area network (CAN) message response times," *Control engineering practice*, vol. 3, no. 8, pp. 1163–1169, 1995.

[51] M. Wilding, "A machine-checked proof of the optimality of a real-time scheduling policy," in *Proceedings of the International Conference on Computer Aided Verification (CAV)*, 1998, pp. 369–378.

[52] F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced CPU energy," in *Proceedings of the 36th IEEE Annual Conference on Foundations of Computer Science*, 1995, pp. 374–382.

[53] Q. Zhou, J. Huang, J. Li, and Z. Li, "Response time analysis for hybrid task sets under fixed priority scheduling," in *Proceedings of the 28th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2022, pp. 108–120.

[54] B. Ziliani, D. Dreyer, N. R. Krishnaswami, A. Nanevski, and V. Vafeiadis, "Mtac: A monad for typed tactic programming in Coq," *Journal of functional programming*, vol. 25, 2015.