Abstract Response-Time Analysis: A Formal Foundation for the Busy-Window Principle

Sergey Bozhko
Max Planck Institute for Software Systems (MPI-SWS)
sbozhko@mpi-sws.org

Björn B. Brandenburg
Max Planck Institute for Software Systems (MPI-SWS)
bbb@mpi-sws.org

Abstract

This paper introduces the first general and rigorous formalization of the classic busy-window principle for uniprocessors. The essence of the principle is identified as a minimal set of generic, high-level hypotheses that allow for a unified and general *abstract response-time analysis*, which is independent of specific scheduling policies, workload models, and preemption policy details. From this abstract core, the paper shows how to obtain concrete analysis instantiations for specific uniprocessor schedulers via a sequence of refinement steps, and provides formally verified response-time bounds for eight common schedulers and workloads, including the widely used fixed-priority (FP) and earliest-deadline first (EDF) scheduling policies in the context of fully, limited-, and non-preemptive sporadic tasks. All definitions and proofs in this paper have been mechanized and verified with the Coq proof assistant, and in fact form the common core and foundation for verified response-time analyses in the Prosa open-source framework for formally proven schedulability analyses.

2012 ACM Subject Classification Computer systems organization → Real-time systems; Software and its engineering → Scheduling; Theory of computation → Scheduling algorithms

Keywords and phrases hard real-time systems, response-time analysis, uniprocessor, busy window, fixed priority, EDF, verification, Coq, Prosa, preemptive, non-preemptive, limited-preemptive

Related Version The conference version of this report appears in *Proceedings of the 32nd Euromicro Conference on Real-Time Systems* (ECRTS 2020).

Supplement Material The Coq proof, a browsable specification, and artifact evaluation instructions can be found at `https://people.mpi-sws.org/~sbozhko/ECRTS20/AbstractRTA.html`.

Funding 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). Funded in part by Deutsche Forschungsgemeinschaft (DFG) — 391919384.

Acknowledgements We thank the members of the ANR-DFG joint project RT-PROOFS, in particular Maxime Lesourd and Sophie Quinton, for their valuable feedback and suggestions.

1 Introduction

The *busy-window principle* is one of the most fundamental and widely known real-time scheduling concepts. The basic idea—to bound a task's *worst-case response time* by analyzing the interval (or "window") during which the processor remains continuously "busy" executing a given task or interfering workload—has been applied in scores of papers on many different scheduling policies, system models, and workload types. In practical terms, the busy-window principle, in the form of *response-time analysis* (RTA) [3, 11, 27, 29, 33], provides the theoretical underpinnings of popular commercial analysis tools (*e.g.*, [25]).

However, despite its undeniable significance, to date no unifying theoretical framework has been presented that explains, and rigorously justifies, why the busy-window principle is applicable in so many diverse settings. Rather, the general idea has become part of the real-time folklore, spread across many papers, where it is frequently re-developed "from scratch" using ad-hoc notation and problem-specific definitions. As previously noted by Bril et al. [7], this results in papers differing in subtle but critical ways, especially when proofs are rooted more in intuition and implicit assumptions than formal derivation.

To place this central element of real-time scheduling theory on firm footings, and to enable a systematic development of verified RTAs in the Prosa [9, 34] open-source framework for formally proven schedulability analyses, we present a rigorous, formally verified proof of the busy-window principle using the Coq proof assistant [41]. Our proof is general and proceeds from first principles, without the use of a "critical instant" argument. At its core is a new *abstract RTA* that is independent of specific scheduling policies, workload models, and preemption details. The existence of this abstract RTA, which relies only on a minimal set of generic, high-level assumptions, explains why the busy-window principle is so widely applicable and clearly identifies its mathematical essence.

From this abstract core, our proof proceeds via a sequence of refinement steps to obtain concrete RTA instantiations for specific schedulers. As practical examples, we provide *formally verified* RTAs for *eight* scheduler and workload combinations: *earliest-deadline first* scheduling of *fully preemptive* tasks (EDF), *fully non-preemptive* tasks (NP-EDF), *segmented limited-preemptive* tasks (LP-EDF), and tasks with *floating non-preemptive sections* (EDF-NPS), as well as *fixed-priority* scheduling of the same task models (denoted FP, NP-FP, LP-FP, and FP-NPS, respectively), all for *arbitrary arrival curves* and *arbitrary deadlines*. Of these, three RTAs are completely novel in that they were not yet derived in prior work (NP-EDF, LP-EDF, and EDF-NPS),[1] and all but one (fully preemptive FP [13]) are formally verified here for the first time. In particular, we provide the first verification of Davis et al.'s [11] revised analysis of the CAN protocol (an instance of NP-FP scheduling).

At a more technical level, this paper makes a contribution in the area of mechanized proofs for real-time systems. Despite many advances in proof assistants in recent years, mechanized proofs still suffer from a key problem: even a conceptually simple change in the underlying model can render a mechanized proof thoroughly inapplicable, and thereby easily induce dozens of person-hours of "proof maintenance." This problem is even more relevant in the area of real-time systems, where a lot of proofs rely on similar ideas but apply to (slightly) different models. It would thus be a Sisyphean task to mechanize the analysis of many such related models without a general underlying theory independent of minor model variations. Abstract RTA provides such a foundational theory for the busy-window principle.

Related Work The origins of the busy-window principle date back more than three decades; a good account of its history is provided by Audsley et al. [2] and Sha et al. [37].

Liu and Layland [30], in their classic analysis of FP scheduling with *rate-monotonic* (RM) priorities, established that a periodic task exhibits its worst-case response time if all higher-priority tasks release jobs simultaneously and at their maximum possible rate, which they called the *critical instant* [30]. More than a decade later, a closer examination of the critical instant led to the independent discovery of exact RTAs for FP scheduling (with any priority order) based on the busy-window principle by several groups of authors [3, 27, 28, 29, 33].

Lehoczky [29] in particular formulated the concept of a "level-i busy period" (a.k.a. busy

[1] Note that our claim is specific to RTAs—there are prior *schedulability analyses* of NP-EDF [26], LP-EDF [4] and EDF-NPS [5, 39], which however do not yield nontrivial response-time bounds (*i.e.* they provide only a "schedulable"/"not schedulable" answer). We focus in this paper exclusively on RTAs.

window), which he defined as "a time interval $[a, b]$ within which jobs of priority i or higher are processed throughout $[a, b]$ but no jobs of level i or higher are processed in $(a - \varepsilon, a)$ or $(b, b + \varepsilon)$ for sufficiently small $\varepsilon > 0$" [29]. Based on this definition, he provided an RTA for tasks with arbitrary deadlines (*i.e.*, relative deadlines independent of a task's period), a case which Joseph and Pandya's earlier RTA [27] failed to account for. Also relying on the busy-window principle (but without giving a precise definition), Audsley et al. [3] and Tindell et al. [42] improved the approach to support release jitter and other workload extensions.

From these beginnings, the busy-window concept spread far and wide (*e.g.*, see [6, 7, 11, 12, 20, 21, 23, 24, 25, 36, 40, 43, 45], to list a few examples across time), and can still be encountered on a regular basis in recent work (*e.g.*, [22, 35, 46]). However, while the busy-window *idea* spread, Lehoczky's *definition* [29] did not. Rather, the concept was reused and adapted in many ways, to the effect that remarkably few papers agree on an *exact* definition of a busy window, or even provide one at all [7].

As a result of this evolution, papers tend to use diverging definitions, notations, and proof strategies, so that it may seem that there are a lot of different methods to obtain a response-time bound, when in fact most papers follow essentially a common argument. Time and again, papers reason about a critical instant or otherwise construct a worst-case execution scenario, infer a recurrence, argue that the result of a fixed-point search implies a response-time bound, define a search space, *etc.* Often this is done in analogy with earlier results and not supported by rigorous proof, which can all too easily lead to misconceptions and flaws—and unfortunately has done so more than once (*e.g.*, [9, 10, 11, 32]).

Another byproduct of the state of the art's paper-by-paper approach is that it obscures commonalities within superficially different proofs. Case in point, EDF and FP historically have been analyzed using quite different and policy-specific terminology. Indeed, the existing RTAs for FP [3, 7, 27, 29, 42, 45] and RTAs for EDF [20, 21, 23, 40] *look* like substantially different analyses. However, as we show in this paper, RTAs for EDF and FP share the same fundamental proof scheme, and can in fact be obtained as instantiations of our abstract RTA.

Bril et al. [7] took an important first step towards rectifying the status quo by introducing the concept of an "active period" as a general, foundational concept that is independent of a particular scheduling algorithm and thus reusable across papers. We argue that this is a vital direction—a new RTA should not start from first principles, but rather build on a well-understood *general* and *formal* foundation that comes with clear and simple proof obligations that justify the application of the busy-window principle.

In contrast to Bril et al.'s "active period" [7], the foundation provided in this paper is backed in full by a formal, mechanized proof checked with the Coq proof assistant. The first successful attempt to mechanize schedulability analysis is due to Wilding [44], who proved optimality of EDF on uniprocessors using the early Nqthm theorem prover. In another early effort more closely related to our work, Dutertre [13] proved the correctness of the classic RTA for preemptive FP scheduling with blocking terms already 20 years ago using the PVS proof system, albeit not for arbitrary arrival curves, nor as part of a general, reusable framework. In the same work, Dutertre described and verified the behavior of the priority-ceiling protocol [38]. Much more recently, Zhang et al. [47] formally specified the priority-inheritance protocol [38] using Isabelle/HOL, and proved a blocking bound for the protocol.

We build in this paper on Prosa [34], a Coq-based framework for real-time scheduling theory that emphasizes readability of the specification, which was introduced by Cerqueira et al. [9] in 2016. In 2018, Fradet et al. [14] presented a result (also using Prosa/Coq) that in principle could be used to obtain RTAs for some of the scheduling policies considered herein, but this possibility was never pursued. In a more applied direction, Fradet et al. [15]

introduced CertiCAN, a Prosa-based tool for the certification of CAN schedulability analysis results. Finally, in particularly impressive recent work, Guo et al. [19] connected Prosa [9, 34] with a real-time extension of CertiKOS [17], thereby obtaining a verified OS kernel with an end-to-end, machine-checked schedulability proof.

Contributions In summary, this paper advances the state of the art in three ways. First, we introduce the first *general* and *rigorous* formalization of the busy-window principle by means of a unified and general *abstract RTA*, which follows from a small, explicit set of *precise* hypotheses.[2] The proposed abstract RTA is intentionally independent of most practical details (such as specific scheduling policies, how and when tasks can be preempted, arrival models, whether they share resources or self-suspend, *etc.*) and can serve as a common basis for the analysis of a wide range of uniprocessor schedulers and workloads.

Second, *all definitions and proofs in this paper have been mechanized and verified* with the Coq proof assistant [41] in the open-source Prosa framework [9, 34]. Moreover, the work presented in this paper has become the common foundation for mechanized uniprocessor RTAs and enabled support for multiple scheduler and workload combinations in the recently released Prosa version 0.4. Our machine-checked proofs ($\approx$ 9,000 lines of code and comments) are fully documented and freely available for reuse and inspection [34]. As part of the artifact evaluation, we provide an overview of our formal proof and cross-reference key results in the paper with the corresponding lemmas and theorems in the formal Coq development [1].

Third, as a case study in applying abstract RTA to specific scheduling problems, we provide eight formally verified response-time analyses for non-self-suspending sporadic tasks with arbitrary deadlines and arbitrary arrival curves under EDF, NP-EDF, LP-EDF, EDF-NPS, FP, NP-FP, LP-FP, and FP-NPS uniprocessor scheduling. For seven of these schedulers, this paper presents *the first formal verification* of a state-of-the-art RTA. In fact, to the best of our knowledge, for three of these policies—NP-EDF, LP-EDF, and EDF-NPS—we provide *the first known RTAs.* For the other five policies, this paper verifies the known exact RTAs. In particular, the four RTAs for FP scheduling verified in this paper correspond to results that can also be obtained from Yao et al.'s general analysis of LP-FP scheduling [45].

2 System Model

In this section, we describe the general system model on which we base our analysis. We focus on unit-speed uniprocessor systems in this paper and assume a discrete-time model, where the smallest quantity $\varepsilon \triangleq 1$ represents an indivisible unit of time (*e.g.*, a processor cycle).

Workload We consider workloads modeled as a set of n sporadic real-time tasks $\tau = \{\tau_1, \ldots, \tau_n\}$. Each task $\tau_i = (C_i, D_i, \alpha_i)$ is characterized by its *worst-case execution time* (or *cost*) C_i, its *relative deadline* D_i, and an *arrival-bound function* $\alpha_i(\Delta)$, which upper-bounds the number of times that τ_i is activated in any interval of length Δ. We also define the *request-bound function* (*RBF*) of task τ_i as $RBF_i(\Delta) \triangleq C_i \times \alpha_i(\Delta)$.

Whenever a task is activated, a corresponding *job* is released. We let $J_{i,j}$ denote the j-th job (or activation) of task τ_i. Each job $J_{i,j}$ has a *release* (or *activation*) time $a_{i,j}$, *absolute deadline* $d_{i,j} = a_{i,j} + D_i$, and *execution time* $c_{i,j}$, where $0 \leq c_{i,j} \leq C_i$. To finish, $J_{i,j}$ must receive exactly $c_{i,j}$ units of service from the scheduler. We denote $J_{i,j}$'s *completion* (or *finish*)

[2] We use the term "hypothesis" in the mathematical sense to refer to an explicitly stated assumption upon which a proof rests, not in the colloquial sense meaning an "unproven theory or conjecture."

time as $f_{i,j}$. A job's cost may be zero, in which case it is trivially finished immediately upon release. A job is *pending* at time t if it is released and not yet completed (*i.e.*, if $a_{i,j} \le t < f_{i,j}$).

A job's *response time* $r_{i,j}$ is given by $r_{i,j} \triangleq f_{i,j} - a_{i,j}$. The goal of a *response-time analysis* of task τ_i is to establish an upper bound R_i such that $r_{i,j} \le R_i$ for any job $J_{i,j}$ of τ_i. Note that such an upper bound need not be exact, nor does it necessarily exist.

Schedule and Service We next define the central notions that relate the workload to the underlying processor model. For clarity, we let $\mathbb{T}$ denote the *time* domain (the natural numbers including zero, *i.e.*, $\mathbb{T} = \mathbb{N}$), $\mathbb{B} = \{0, 1\}$ the boolean domain, and $\mathbb{J} = \{J_{i,j}\}_{\forall i,j}$ the set of all jobs. Furthermore, we let $\mathbb{S}$ denote the *service* domain, which is another synonym of the natural numbers (*i.e.*, $\mathbb{S} = \mathbb{N}$) that we use to disambiguate the notions of an instant or duration $t \in \mathbb{T}$ and the amount of accumulated service $\rho \in \mathbb{S}$ received by a job.

A *schedule* is a function $\sigma : \mathbb{T} \to \{\perp\} \cup \mathbb{J}$ that maps each point in time t to the job (if any) that is scheduled at time t, or to a constant $\perp$ that indicates that the processor is idle. A job can be scheduled only when it is pending.

A *scheduling policy* determines how the schedule is constructed. In Sections 3–6.1, we place no restriction on the type of scheduling policy considered. In Section 6.2, we focus our attention on *job-level fixed-priority* (JLFP) policies [8], which are policies that assign a fixed priority to each pending job, and then execute (any one of) the job(s) with maximal priority. Finally, in Section 7, we instantiate our abstract RTA for two specific JLFP policies, namely the two most prominent representatives of this class: FP and EDF scheduling. We let $J_k \succeq J_i$ denote that J_k has a priority that is higher than or equal to J_i's priority, and let $J_k \succ J_i$ denote that J_k has priority strictly higher than J_i.

The scheduler is assumed to be work-conserving, which we define precisely in Section 3.2. We further consider all scheduling overheads to be negligible, or equivalently, to already be included in each job's cost $c_{i,j}$. While our framework and proof strategy do not preclude the explicit modeling of overheads, a detailed consideration of overhead accounting issues is beyond the scope of this paper and left to future work.

As we assume a unit-speed uniprocessor, a job $J_{i,j}$ receives one unit of service at time t iff $\sigma(t) = J_{i,j}$. The *cumulative service* of a job $J_{i,j}$ received within a time interval $[t_1, t_2)$, denoted $serv_\sigma(J_{i,j}, [t_1, t_2))$, is given by $serv_\sigma(J_{i,j}, [t_1, t_2)) \triangleq |\{t \mid t \in [t_1, t_2) \wedge \sigma(t) = J_{i,j}\}|$. We also rely on the notions of the *total service* received up to time t, denoted $serv_\sigma(J_{i,j}, t) \triangleq serv_\sigma(J_{i,j}, [0, t))$, and the service received by task τ_i in time interval $[t_1, t_2)$, which is the total service received by jobs of τ_i in the given interval: $serv_\sigma^{\tau_i}([t_1, t_2)) \triangleq \sum_j serv_\sigma(J_{i,j}, [t_1, t_2))$.

Arrivals and Workload For notational clarity, we introduce the *arrival sequence* $a(t) \triangleq \{J_{i,j} \mid \forall i, j : a_{i,j} = t\}$, which is a function mapping each time t to the (possibly empty) set of jobs released at time t. The *workload* of jobs of task τ_i in a given time interval $[t_1, t_2)$ is the cumulative cost of all jobs of τ_i released in that interval: $wl^{\tau_i}([t_1, t_2)) \triangleq \sum \{c_{i,j} \mid \forall j : t_1 \le a_{i,j} < t_2\}$. Similarly, $wl([t_1, t_2)) \triangleq \sum_{i=1}^{n} wl^{\tau_i}([t_1, t_2))$ is the workload of all jobs. (We write $\sum \{F(x) \mid \forall x : P(x)\}$ to denote $\sum_{x \in \{x \mid P(x)\}} F(x)$, for any F and P.)

Preemption Model A job $J_{i,j}$ is represented by a sequence of $q_{i,j}$ non-preemptive segments $c_{i,j,k}$ such that $\sum_{k=1}^{q_{i,j}} c_{i,j,k} = c_{i,j}$. (In case of a fully preemptive job, each segment is simply of length ε, the smallest discrete quantity of time.) We denote the longest segment of job $J_{i,j}$ as $c_{i,j}^{max} = \max\{c_{i,j,k}\}_{1 \le k \le q_{i,j}}$, and correspondingly let NPS_i denote the *maximum non-preemptive segment length* of task τ_i, such that $c_{i,j}^{max} \le NPS_i$ for any $J_{i,j}$.

We denote a job $J_{i,j}$'s last segment as $c_{i,j}^{last} = c_{i,j,(q_{i,j})}$. Importantly, once $J_{i,j}$ has received enough service to start $c_{i,j}^{last}$, it cannot be preempted until it completes. We call this amount of service the job's *run-to-completion threshold* $rct_{i,j} \triangleq c_{i,j} - c_{i,j}^{last} + \varepsilon$. Correspondingly, task τ_i's *run-to-completion bound* is a constant RCT_i such that $rct_{i,j} \leq RCT_i$ for each $J_{i,j}$. That is, once a job of task τ_i has received at least RCT_i units of service, it is guaranteed to have reached its last non-preemptive segment and thus to complete without further preemptions.

3 High-Level Overview and Abstract Foundation

We begin with a high-level overview of our approach and then introduce the foundation upon which the analysis rests, namely the definitions and hypotheses that form the abstract RTA.

We employ one of the most commonly used tools in computer science, namely *reductions*. Reductions play a central role in computer science as they allow transferring a solution for a fundamental problem to solve another, often more concrete problem. For example, recall the Boolean Satisfiability Problem (SAT), which asks to determine whether a given formula has a satisfying assignment. Now if one faces another, domain-specific problem A, as long as one has an algorithm that decides SAT (*e.g.*, a SAT-solver) *and* a reduction that maps an instance of problem A to an equivalent instance of problem SAT, one can decide problem A by applying the reduction from A to SAT and then running a SAT-solver.

In this work, we use the conceptually same idea, but do not reduce to SAT. Rather, we define a small *abstract model*, which does not have all the intricacies of a conventional scheduling model, but which nevertheless allows us to state the problem of finding a response-time bound. If the resulting abstract model correctly captures the essential properties that are necessary for an RTA via the busy-window principle, then we can expect that a response-time bound for the abstract model can be transferred to a given concrete model via a sequence of reductions from the concrete to the abstract model.

As shown in Figure 1, the central result of this paper is the abstract RTA theorem (Theorem 18), which solves the RTA problem for the abstract model (Section 3.1). On top of the abstract model, we provide a sequence of model refinements along with proofs of correctness for the corresponding reductions (in Figure 1, the refinement sequence proceeds in reverse direction of the arrows). In particular, by assuming that tasks are sequential, it is possible to obtain a more accurate bound on interference (Section 5). From this common base, we then obtain general EDF and FP models (Theorems 31 and 32), which however still abstract from specific preemption models (Section 6). Ultimately, the refinement sequence connects the abstract model to concrete models; conversely, the reduction sequence (indicated by the arrows in Figure 1) yields RTAs for concrete models (Section 7).

As indicated in Figure 1, we split reductions along two orthogonal concerns:

- the *scheduling policy* — how concurrently pending jobs are sequenced; and
- the *preemption model* — when is it possible for the scheduler to change the executing job.

Indeed, both the scheduling policy and preemption model are essentially just parameters of the underlying abstractions. As they are independent of each other, we can construct reductions by combining available scheduling models and preemption policies, which we have done in this paper for all combinations of the EDF and FP scheduling policies and four preemption models, as illustrated in Figure 1 and discussed in Section 7.

Importantly, our proof framework is flexible in the sense that one has the freedom to expand the graph of reductions depicted in Figure 1 at any node. For instance, in future work it will be possible to add branches for FIFO scheduling (*e.g.*, as a sibling to Theorems 31 and 32), another preemption model (*e.g.*, scheduling with preemption thresholds), other task

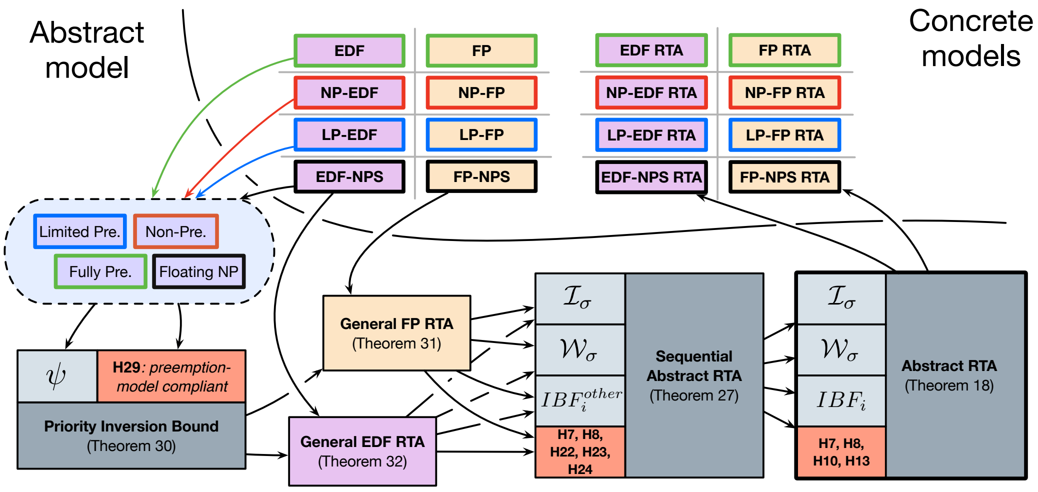

Figure 1 Structure of the presented abstract RTA framework. Black arrows indicate reduction steps from concrete to abstract models. For each reduction (*e.g.*, NP-EDF RTA $\mapsto$ abstract RTA), concrete definitions of the interface functions must be instantiated and all abstract analysis hypotheses must be proven to hold for the specific preemption model (*e.g.*, NP) and scheduler (*e.g.*, EDF).

models (*e.g.*, self-suspending tasks), or support for real-time locking protocols.

3.1 Abstract Model

At the core of the proposed framework lies the abstract model, which rests on four central hypotheses. Intuitively speaking, we assume that the processor is not overloaded (Hypothesis 7), that the scheduler is work-conserving (Hypothesis 8), that worst-case interference is bounded (Hypothesis 10), and that a solution to the response-time recurrence is known (Hypothesis 13). We define these assumptions precisely in Sections 3.4, 3.5, and 3.7 below.

▶ **Definition 1.** *An* abstract model *is a tuple* $(\tau_i, \mathcal{I}_\sigma, \mathcal{W}_\sigma, IBF_i)$*, where* τ_i *is the task under analysis,* $\mathcal{I}_\sigma$ *is the (abstract)* interference *function,* $\mathcal{W}_\sigma$ *is the (abstract)* interfering workload *function,* IBF_i *is the* interference-bound function*, and where* $\mathcal{I}_\sigma$*,* $\mathcal{W}_\sigma$*, and* IBF_i *satisfy the requirements imposed in Hypotheses 7, 8, 10, and 13.*

The three functions describe the evolution of an (abstract) system and thereby form the *interface* of abstract RTA. Specifically, reducing a concrete model to the abstract model means **(i)** *instantiating* $\mathcal{I}_\sigma$, $\mathcal{W}_\sigma$, and IBF_i such that they capture the concrete model's semantics, and **(ii)** *proving* that the chosen definitions satisfy Hypotheses 7, 8, 10, and 13.

Given an abstract model $(\tau_i, \mathcal{I}_\sigma, \mathcal{W}_\sigma, IBF_i)$, the problem of bounding the worst-case response time of τ_i is defined as follows.

▶ **Definition 2** (abstract RTA problem)**.** *Given a constant* R*, decide whether the inequality* $f_{i,j} - a_{i,j} \le R$ *holds for every job* $J_{i,j}$ *of task* τ_i*, for any arrival sequence* $a(t)$ *and schedule* $\sigma(t)$ *consistent with* $(\tau_i, \mathcal{I}_\sigma, \mathcal{W}_\sigma, IBF_i)$ *in the sense of Hypotheses 7, 8, 10, and 13.*

We emphasize that we place no restrictions on $a(t)$ and $\sigma(t)$ other than the (arguably quite weak) restrictions imposed by the task model and Hypotheses 7, 8, 10, and 13. In particular, note that Definitions 1 and 2 are silent on matters of scheduling policy, preemption model, task-model specifics, *etc.*, which are all abstracted by the interface functions, as discussed next.

3.2 Abstract Interference

A job's execution may be postponed by the environment and/or the system due to many different factors such as preemption by higher-priority jobs, non-preemptive lower-priority jobs, jitter, black-out periods in hierarchical scheduling, *etc.* We collectively refer to *any* such delay as (*abstract*) *interference.* In particular, we consider the execution of other jobs of the same task, frequently called self-interference in prior work, to also constitute interference.

Formally speaking, interference is a function $\mathcal{I}_\sigma : \mathbb{J} \times \mathbb{T} \to \mathbb{B}$ that satisfies Hypotheses 7, 8, 10, and 13. Intuitively, $\mathcal{I}_\sigma$ is a predicate that determines, in the context of a given schedule σ, for any job $J_{i,j}$ and any time t, whether $J_{i,j}$ *could* execute at time t *if it were pending* at time t. Note that a job does not have to be *actually* pending to experience (abstract) interference. This seemingly unnatural definition will turn out to be useful when bounding the response time of jobs that arrive after the beginning of a busy window (Hypothesis 10). To illustrate the idea, we provide a simple example; further examples can be found in Section 7.

▶ **Example 3.** For a fully preemptive JLFP model assuming Liu & Layland [30] tasks (*i.e.*, no self-suspensions or shared resources), $\mathcal{I}_\sigma$ can be defined as follows:

$$\mathcal{I}_\sigma(J_{i,j}, t) \triangleq \exists\, J_{h,k} :\ J_{h,k} \neq J_{i,j} \ \wedge\ \sigma(t) = J_{h,k} \ \wedge\ J_{h,k} \succeq J_{i,j}.$$

That is, interference occurs if another job with higher or equal priority is scheduled. See Figure 2 for an illustration. In Section 7, we later provide a similar instantiation that also works in the presence of non-preemptive sections.

In general, conceptually similar interference functions can be defined for many schedulers and workload models, but finding the most appropriate definition can require some ingenuity in case of complex workloads or intricate scheduling policies.

Building on the interference function $\mathcal{I}_\sigma$, we define the *cumulative interference* $C_\mathcal{I}$ of a job $J_{i,j}$ within a time interval $[t_1, t_2)$ as $C_\mathcal{I}(J_{i,j}, [t_1, t_2)) \triangleq \sum_{t=t_1}^{t_2-1} [\![\mathcal{I}_\sigma(J_{i,j}, t)]\!]_\mathbb{1}$, where $[\![x]\!]_\mathbb{1}$ denotes the *indicator function* that evaluates to 1 if x is true, and 0 otherwise.

3.3 Abstract Interfering Workload

The second function of the abstract interface, called *interfering workload* $\mathcal{W}_\sigma$, is of a more technical nature and is intended to describe the *potential for future interference*, in the sense that it allows "foreseeing" (in the context of a fixed schedule σ) the amount of interference that a job can incur in the future. For example, the release of a higher-priority job with cost $c_{i,j}$ means that a lower-priority job may subsequently suffer $c_{i,j}$ units of interference.

Formally, the interfering workload $\mathcal{W}_\sigma : \mathbb{J} \times \mathbb{T} \to \mathbb{N}$ is a function that satisfies Hypotheses 7, 8, 10, and 13. Intuitively, it is useful to think of the function as indicating, for any job $J_{i,j}$ and any time t, the amount of potential interference for job $J_{i,j}$ that is introduced into the system at time t in a given schedule σ. This idea will become clearer with an example.

▶ **Example 4.** Continuing Example 3, an appropriate *interfering workload* function for a fully preemptive JLFP scheduling model is given by

$$\mathcal{W}_\sigma(J_{i,j}, t) \triangleq \sum \{c_{h,k} \ \mid \forall J_{h,k} \in a(t) \setminus \{J_{i,j}\} : J_{h,k} \succeq J_{i,j}\}\,.$$

That is, the interfering workload introduced at time t is simply the total cost of all higher- or equal-priority jobs released at t (excluding job $J_{i,j}$ itself). See Figure 2 for an illustration.

While $\mathcal{W}_\sigma$ may appear to be a somewhat unfamiliar concept on first sight, its essential role will become fully apparent next with the introduction of a general notion of "busy window."

Analogously to cumulative interference, we define the *cumulative interfering workload* $C_\mathcal{W}$ of a job $J_{i,j}$ within a time interval $[t_1, t_2)$ as the aggregation of $\mathcal{W}_\sigma$ across said interval.

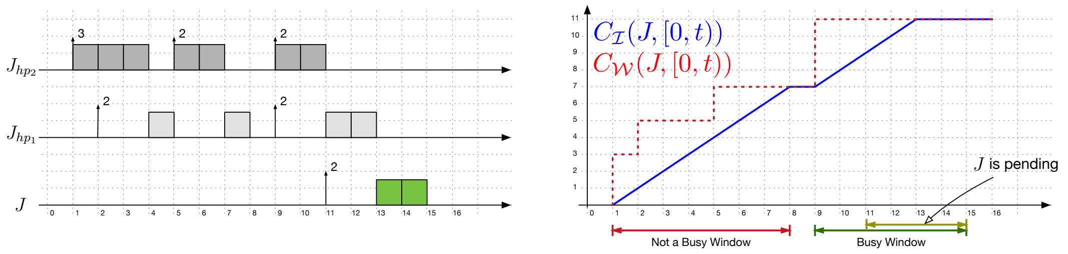

Figure 2 Illustration of an abstract busy window. Let J, J_{hp_1}, J_{hp_2} be arbitrary jobs such that $J_{hp_2} \succeq J_{hp_1} \succeq J$. The dashed red curve shows $C_{\mathcal{W}}(J, [0, t))$, and the solid blue curve shows $C_{\mathcal{I}}(J, [0, t))$, with $\mathcal{I}_\sigma$ and $\mathcal{W}_\sigma$ defined as in Examples 3 and 4. Intuitively, the difference $C_{\mathcal{W}}(J, [0, t)) - C_{\mathcal{I}}(J, [0, t))$ expresses the amount of "pending" interference that can defer the execution of job J.

3.4 Abstract Busy Window

As motivated in Section 1, the ultimate goal of this paper is to establish a general and reusable formal foundation for the busy-window principle, which necessarily requires a general and precise definition of the concept of a "busy window" (or, interchangeably, "busy interval"). While it is (deceptively) easy to state the *intuitive* idea of a busy window—the processor is continuously "busy" executing a job under analysis or some interfering workload—and while it is also usually not too difficult to define this idea for a given *specific* model, coming up with a good definition in the general case is another matter entirely.

In fact, this challenge is the motivation behind the choice of abstract interface functions $\mathcal{I}_\sigma$ and $\mathcal{W}_\sigma$. Using these two functions, we can define a single, general notion of a "busy interval" that is independent of any specific model, and which thus can capture the essence of the busy-window idea even across very different scheduler and workload models.

The key insight is that the workload function $\mathcal{W}_\sigma$ *produces* interfering workload, while the interference function $\mathcal{I}_\sigma$ *consumes* such workload. Thus, when $C_{\mathcal{W}}$ equals $C_{\mathcal{I}}$, there is no more interfering workload to process, so the processor necessarily becomes "quiet" and the corresponding busy interval ends. This observation leads us to the following definition.

▶ **Definition 5** (quiet time). *A point in time t is a quiet time w.r.t. a job $J_{i,j}$ if*

$$C_{\mathcal{I}}(J_{i,j}, [0, t)) = C_{\mathcal{W}}(J_{i,j}, [0, t)) \wedge (t \leq a_{i,j} \vee f_{i,j} \leq t).$$

In other words, at a quiet time t, we require the cumulative interference up to time t to be equal to the cumulative interfering workload, which indicates that the potential interference seen so far has been fully "consumed" (*i.e.*, no more higher-priority work or other kinds of delay are pending). Furthermore, to ensure that a job $J_{i,j}$'s busy interval (defined next) actually captures its execution, we require that $J_{i,j}$ cannot *both* be pending before the quiet time t *and* also at time t (*i.e.*, $\neg(a_{i,j} < t \wedge t < f_{i,j}) \Leftrightarrow t \leq a_{i,j} \vee f_{i,j} \leq t$). Thus:

▶ **Definition 6** (busy interval). *An interval $[t_1, t_2)$ is a busy interval w.r.t. job $J_{i,j}$ if* ***(i)*** *$a_{i,j} \in [t_1, t_2)$,* ***(ii)*** *t_1 is a quiet time* ***(iii)*** *t_2 is a quiet time, and* ***(iv)*** *no $t \in (t_1, t_2)$ is a quiet time.*

In other words, we say that a given interval is a job's busy interval if the interval contains the arrival of the job, starts with a quiet time, and remains non-quiet until it ends with a quiet time. Figure 2 illustrates the busy-window concept for a fully preemptive JLFP model.

Note that it follows from Definition 6 that a job's busy window, if it exists, is unique (which we formally establish in our Coq proof). It also bears repeating that Definition 6 applies to a specific job as this simplifies the formal Coq development, whereas prior work traditionally defines the notion of a busy window w.r.t. a task or priority level [29].

With the abstract notion of a busy interval in place, we can finally state our first two hypotheses, which constrain $\mathcal{I}_\sigma$ and $\mathcal{W}_\sigma$ via Definitions 5 and 6. First, obviously a response-time bound can exist only if busy windows are of finite length.

▶ **Hypothesis 7.** *Busy intervals are bounded by a constant L*: for any $J_{i,j}$, there is a busy interval $[t_1, t_2)$ w.r.t. $J_{i,j}$ and $t_2 - t_1 < L$.

Clearly, busy intervals are not actually always bounded; for example, if the processor is overloaded. However, no response-time analysis is applicable to workloads that exhibit unbounded busy intervals. Hypothesis 7 thus must be checked and proven to be satisfied in order for abstract RTA to be applicable. In fact, this hypothesis is analogous to the common requirement that a task set's total utilization cannot exceed 100%. However, Hypothesis 7 is more general since it reflects also any other abstract interference factors modeled by $\mathcal{I}_\sigma$.

Next, we require "work conservation" in an abstract sense.

▶ **Hypothesis 8.** The scheduler is *work-conserving in the abstract sense*: for any job $J_{i,j}$, its busy interval $[t_1, t_2)$, and any point in time $t \in [t_1, t_2)$, $J_{i,j}$ incurs interference at time t iff $J_{i,j}$ is not scheduled at time t: $\mathcal{I}_\sigma(J_{i,j}, t) \iff \sigma(t) \neq J_{i,j}$.

Intuitively, Hypothesis 8 requires the abstract interference predicate to describe the interference "correctly"—the scheduler must either schedule $J_{i,j}$, or there is something else that interferes with $J_{i,j}$, which matches the intuitive understanding of "work conservation." Note that we consider only time instants within a busy interval; we can thus be sure that there is some pending interference in the system (otherwise t would be a quiet time).

However, it is also interesting to note that Hypothesis 8 does *not* state whether the processor idles at time t (*i.e.*, $\sigma(t) = \bot$ is permissible under Hypothesis 8 if $\mathcal{I}_\sigma(J_{i,j}, t)$ holds), which is necessary for generality: otherwise, abstract RTA would not be applicable to workload models with self-suspensions, release jitter, or delayed budget replenishments.

3.5 Abstract Interference Bound Function

The *interference-bound function* $\mathit{IBF}_i : \mathbb{T} \times \mathbb{T} \to \mathbb{T}$ is the third and last element of the behavioral interface of our analysis. The notion of an IBF as a bound on interference during some interval is well-known from prior work on the analysis of specific schedulers. However, to obtain the generality needed for abstract RTA, our definition differs from prior work in that it explicitly considers the *relative offset* of the job under analysis within its busy interval.

▶ **Definition 9.** *Given a job $J_{i,j}$ and its busy interval $[t_1, t_2)$, we call $A \triangleq a_{i,j} - t_1$ the* offset *(or* relative arrival time*) of job $J_{i,j}$ w.r.t. the beginning of the busy interval t_1.*

Based on this notion, abstract RTA requires the existence of a function $\mathit{IBF}_i(A, \Delta)$ that yields a bound on the maximum interference incurred by any job $J_{i,j}$ of task τ_i during an interval of length Δ starting with $J_{i,j}$'s busy window *assuming $J_{i,j}$ has a relative offset A.*

▶ **Hypothesis 10.** Given any job $J_{i,j}$ of a task τ_i and a schedule σ, $J_{i,j}$'s busy interval $[t_1, t_2)$ in σ, and a subinterval $[t_1, t_1 + \Delta) \subseteq [t_1, t_2)$, $\mathit{IBF}_i(a_{i,j} - t_1,\ \Delta)$ bounds the cumulative interference incurred by $J_{i,j}$ during $[t_1, t_1 + \Delta)$: $C_{\mathcal{I}}(J_{i,j}, [t_1, t_1 + \Delta)) \leq \mathit{IBF}_i(a_{i,j} - t_1,\ \Delta)$.

Note that IBF_i bounds the total interference within an interval starting at time t_1 regardless of the job's arrival time. That is, the interval starts from the beginning of the job's busy interval, and not from its arrival time. The reason to add the job's relative offset as a parameter of IBF_i is that it extends the set of possible interference bounds, which is important when analyzing EDF scheduling, where a job's priority depends on its arrival time. To illustrate the idea, consider the following example assuming FP scheduling, where we write $\tau_h \succeq \tau_i$ to indicate that task τ_h has priority no lower than task τ_i.

▶ **Example 11.** For a fully preemptive FP model, IBF_i can be defined as follows:

$$\mathit{IBF}_i(A, \Delta) \triangleq \left(\sum \{\mathit{RBF}_h(\Delta) \mid \forall \tau_h : \tau_h \succeq \tau_i\}\right) - X,$$

where $X = C_i$ if $\Delta > A$, and $X = 0$ otherwise. The term X is subtracted to exclude the cost of the job under analysis itself. Note that IBF_i considers as interference even jobs of task τ_i that arrive *later* than A time units after the beginning of a busy interval (*i.e.*, *after* the job under analysis arrives). While this is required in the most general case (*e.g.*, for non-sequential tasks models), it is overly pessimistic in the context of sequential tasks. In Section 5, we define a refined model for sequential tasks to eliminate this pessimism.

To clarify, both $\mathcal{W}_\sigma$ and IBF_i bound the interference of a given job. However, $\mathcal{W}_\sigma$ is primarily a modeling construct (for defining abstract busy windows) that yields a bound depending on a *specific* job arrival sequence and schedule σ, whereas IBF_i is an analysis tool that bounds interference in *any* schedule σ (compliant with the task model).

3.6 Abstract Response-Time Bound

Using IBF_i, we can proceed to state the actual response-time bound. Consider a job $J_{i,j}$ that arrives A time units after the beginning of its busy window $[t_1, t_2)$. Following a line of reasoning similar to those given by Davis et al. [11] and Yao et al. [45], we use the fact that once $J_{i,j}$ receives $rct_{i,j} \leq RCT_i$ units of service, it becomes non-preemptive and runs to completion, requiring no more than $c_{i,j}^{last} - \varepsilon \leq C_i - RCT_i$ additional units of service. The central question thus is: by when will $J_{i,j}$ receive RCT_i units of service at the latest?

By Hypothesis 8, $J_{i,j}$ receives service whenever it does not incur interference. Thus, if we find some $F \geq 0$ such that $A + F = RCT_i + \mathit{IBF}_i(A,\ A + F)$, then we can conclude that $J_{i,j}$ has received *at least* RCT_i units of service by time $t_1 + A + F$ (or has already completed), and thus will complete no later than by time $t_1 + A + F + (C_i - RCT_i)$. Since $f_{i,j} \leq t_1 + A + F + (C_i - RCT_i)$, and by assumption $a_{i,j} = t_1 + A$, we have $r_{i,j} \leq F + (C_i - RCT_i)$.

However, since $J_{i,j}$'s relative arrival time A is unknown in general, this inequality does not immediately yield a useful bound. Nonetheless, as a first step, we observe that the solution F that is maximal for all $A \in \mathbb{T}$ implies a response-time bound. More precisely, a constant R is a response-time bound for τ_i, if for each $A \in \mathbb{T}$ there exists an F such that

$$A + F = RCT_i + \mathit{IBF}_i(A,\ A + F) \quad \text{and} \quad F + (C_i - RCT_i) \leq R. \tag{1}$$

3.7 Finite Search Space

Equation (1) has no bound on the set of possible offsets (*i.e.*, $A \in \mathbb{T}$), thus while being correct, it is not practical in the sense that it cannot be used to actually compute such an R as that would require enumerating all $A \in \mathbb{T}$. Therefore, to obtain a foundation for a practical analysis, we must restrict Equation (1) to a finite search space. In this section, we focus on the high-level idea underlying the abstract search space, and defer the proof that supports this reasoning to Section 4.2.

First, given Hypothesis 7, it is easy to see that no job of task τ_i has a relative arrival time $A \geq L$ as there is no busy interval longer than L. However, simply limiting the maximum magnitude of A is insufficient since L can be very large (*i.e.*, enumerating every $A < L$ is impractical, given billions of processor cycles per second).

To obtain a sparse search space, it is useful to note that in Equation (1) IBF_i is the only varying term dependent on A. The function IBF_i in turn depends on only two parameters, A and Δ. We can thus ignore any relative offset A for which IBF_i "does not change." Conversely, and more precisely, we say that an offset $A > 0$ is *in the search space* if the partially applied function $\mathit{IBF}_i(A - \varepsilon, \cdot)$ is not equivalent to the partially applied function $\mathit{IBF}_i(A, \cdot)$, that is, if there exists a $\Delta < L$ such that $\mathit{IBF}_i(A - \varepsilon, \Delta) \neq \mathit{IBF}_i(A, \Delta)$.

Finally, as a base case, we always include $A = 0$, which yields the following search space.

▶ **Definition 12.** *The* search space *of task τ_i is given by*

$$\mathcal{A}_i \triangleq \{0\} \cup \left(\bigcup \{A \mid 0 < A < L \ \wedge\ \exists \Delta,\ \mathit{IBF}_i(A - \varepsilon,\ \Delta) \neq \mathit{IBF}_i(A,\ \Delta)\} \right).$$

This leads us to the final hypothesis upon which abstract RTA rests, namely the assumption that a bound R satisfying Equation (1) is known for all relevant offsets (*i.e.*, for any $A \in \mathcal{A}_i$).

▶ **Hypothesis 13.** There exists a constant R such that $\forall A \in \mathcal{A}_i, \exists F,$

$$A + F = \mathit{RCT}_i + \mathit{IBF}_i(A,\ A + F) \text{ and } F + (C_i - \mathit{RCT}_i) \leq R. \tag{2}$$

In more traditional terms, we assume that R is an upper bound on the fixed point of the response-time recurrence for each "relevant" release offset A. Note that we intentionally do not specify how such a constant R is obtained (*i.e.*, by classic fixed-point iteration, exhaustive search, querying some oracle such as a linear-program solver, or by any other means). To establish that abstract RTA is sound (*i.e.*, to show that R is indeed a response-time bound), it is sufficient to know that R bounds each fixed point, regardless of how it was found.

4 Abstract Response-Time Analysis

In this section, we present key proofs that are essential to understanding our analysis. The core of the analysis consists of three theorems: Theorem 15 relates the service received by a job and an interference bound, Theorem 17 proves the reduced search space to be sufficient, and finally Theorem 18 joins the prior two theorems to obtain the abstract response-time bound. The proofs in this section rely solely on Hypotheses 7, 8, 10, and 13 and the definitions in Section 2 (*i.e.*, no additional implicit assumptions are made, which is verified by our Coq proof).

4.1 Service Theorem

Consider an arbitrary job $J_{i,j}$ and its busy interval $[t_1, t_2)$. We begin by observing that, within any time interval $[t, t + \Delta) \subseteq [t_1, t_2)$, the sum of the cumulative service received by $J_{i,j}$ and the cumulative interference experienced by $J_{i,j}$ equals Δ.

▶ **Lemma 14.** *In any time interval $[t, t + \Delta) \subseteq [t_1, t_2)$:*

$$\mathit{serv}_\sigma(J_{i,j}, [t, t + \Delta)) + C_\mathcal{I}(J_{i,j}, [t, t + \Delta)) = \Delta.$$

Proof. At any instant $t \in [t, t + \Delta)$, $J_{i,j}$ either receives a unit of service or incurs one unit of interference (but not both) since the scheduler is work-conserving (Hypothesis 8). From the definitions of cumulative service (Section 2) and cumulative interference (Section 3.2), it follows that the left-hand side is a sum of Δ terms equal to one, with the rest being zero. ◀

Next, note that the service received by $J_{i,j}$ outside its busy interval $[t_1, t_2)$ is equal to zero (which follows from the definition of busy interval), and that a job must be completed by the end of its busy interval. Thus, $J_{i,j}$ receives $c_{i,j}$ units of service during $[t_1, t_2)$, which implies that, for every $\rho \in [0, c_{i,j}]$, there exists a Δ such that $serv_\sigma(J_{i,j}, [t_1, t_1 + \Delta)) = \rho$.

Put differently, a bound of the form $\rho \leq serv_\sigma(J_{i,j}, [t_1, t_1 + \Delta))$ states that $J_{i,j}$ has received *at least* ρ units of service by time $t_1 + \Delta$. The main service theorem, which we establish next, serves to relate such a lower bound on cumulative service to the cumulative interference incurred by $J_{i,j}$ so far. More precisely, if the sum of ρ and the cumulative interference experienced by $J_{i,j}$ within the interval $[t_1, t_1 + \Delta)$ does not exceed Δ, then it follows that $J_{i,j}$ has indeed received at least ρ units of service by time $t_1 + \Delta$.

▶ **Theorem 15.** *For any $\rho \in [0, c_{i,j}]$ and Δ, if $\rho + C_{\mathcal{I}}(J_{i,j}, [t_1, t_1 + \Delta)) \leq \Delta$, then*

$$\rho \leq serv_\sigma(J_{i,j}, [t_1, t_1 + \Delta)).$$

Proof. By Lemma 14, $\Delta = serv_\sigma(J_{i,j}, [t_1, t_1 + \Delta)) + C_{\mathcal{I}}(J_{i,j}, [t_1, t_1 + \Delta))$, and from the premise we have $\rho + C_{\mathcal{I}}(J_{i,j}, [t_1, t_1 + \Delta)) \leq \Delta$, which implies $\rho \leq serv_\sigma(J_{i,j}, [t_1, t_1 + \Delta))$. ◀

This simple but important fact shows that we automatically gain a lower bound on the total amount of service received by a job whenever we upper-bound the cumulative abstract interference that it has incurred. Theorem 15 will be the main tool for proving the abstract RTA theorem (Theorem 18).

4.2 Reduction of the Search Space

As discussed in Section 3.7, the notion of an explicit search space serves to shrink the set of equations that must be solved to obtain a safe response-time bound for a task. While we have already stated and *intuitively* explained the search space in Section 3.7, we now provide a more rigorous argument.

Our method to justify the reduced search space differs from the approaches employed in prior papers, which pursue roughly the following argument: in certain situations, it is possible to transform a schedule σ into an equivalent schedule σ', where the job under analysis arrives one time unit earlier (*i.e.*, $a'_{i,j} = a_{i,j} - \varepsilon$), so that it is sufficient to consider only σ'. That is, one can “move” the arrival of a critical job “to the left” until a schedule belonging to the search space is found. However, while this sketch is intuitively appealing, it is difficult to find a simple argument for why the schedule transformation is correct even for *specific* models, let alone in the abstract general case. Our approach instead automates the reduction procedure based on a more rigorous (but less visual) argument.

Consider two offsets A_1 and A_2: we say that IBF_i is *equivalent* on these offsets if $IBF_i(A_1, \Delta) = IBF_i(A_2, \Delta)$ for any $\Delta < L$, which we denote as $IBF_i(A_1, \cdot) \equiv IBF_i(A_2, \cdot)$.

In the following, we prove that, if $IBF_i(A_1, \cdot) \equiv IBF_i(A_2, \cdot)$, then there is no point in keeping both A_1 and A_2 in the search space. To this end, we show how to transform the equation $A_1 + F_1 = RCT + IBF_i(A_1, A_1 + F_1)$ into the equation $A_2 + F_2 = RCT_i + IBF_i(A_2, A_2 + F_2)$ such that the solution F_1 can be easily computed from the solution F_2.

First, we show that, for *any* offset A, there is an offset A_s in the search space $\mathcal{A}_i$ such that IBF_i is equivalent on A and A_s.

▶ **Lemma 16.** *For any $A < L$, $\exists A_s \in \mathcal{A}_i$ such that $A_s \leq A$ and $IBF_i(A_s, \cdot) \equiv IBF_i(A, \cdot)$.*

Proof. By induction on A. The *base case* $A = 0$ is trivial since 0 is in the search space.

Induction step: given that the property holds for A, we need to prove the claim for $A + \varepsilon$. Consider two cases.

Case 1: $\mathit{IBF}_i(A,\cdot) \equiv \mathit{IBF}_i(A+\varepsilon,\cdot)$. By the induction hypothesis, there is an $A_s \in \mathcal{A}_i$ such that $A_s \leq A < A+\varepsilon$. Moreover, we have $\mathit{IBF}_i(A_s,\cdot) \equiv \mathit{IBF}_i(A,\cdot) \equiv \mathit{IBF}_i(A+\varepsilon,\cdot)$ by transitivity. Hence A_s satisfies the claim also for $A+\varepsilon$.

Case 2: $\mathit{IBF}_i(A,\cdot) \not\equiv \mathit{IBF}_i(A+\varepsilon,\cdot)$. In this case, $A+\varepsilon$ is in the search space itself, because the fact that IBF_i is not equivalent on A and $A+\varepsilon$ implies that there exists a $\Delta < L$ such that $\mathit{IBF}_i((A+\varepsilon)-\varepsilon,\Delta) \neq \mathit{IBF}_i(A+\varepsilon,\Delta)$, which matches the criterion for inclusion in the search space. ◀

However, since the second parameter of $\mathit{IBF}_i(A, A+F)$ in Equation (2) also depends on A, Lemma 16 by itself does not yet allow us to substitute any arbitrary offset with an offset from the search space. Rather, we also need to transform the solution F. The following theorem justifies the replacement.

▶ **Theorem 17.** *Let $A_s \in \mathcal{A}_i$ and let F_s be the corresponding solution such that $A_s + F_s < L$ and $A_s + F_s = \mathit{RCT}_i + \mathit{IBF}_i(A_s, A_s+F_s)$. Then, for any offset $A \in [A_s, A_s+F_s]$ such that $\mathit{IBF}_i(A_s,\cdot) \equiv \mathit{IBF}_i(A,\cdot)$, there exists a solution F such that the following conditions are met: **(i)** $A_s + F_s = A + F$, **(ii)** $F \leq F_s$, and **(iii)** $A+F = \mathit{RCT}_i + \mathit{IBF}_i(A, A+F)$.*

Proof. It is easy to verify that for $F \triangleq A_s + F_s - A$ the conclusion of the theorem holds.

(i) The equality $A_s + F_s = A + F$ trivially holds.

(ii) From (i), by moving the term A_s to the right-hand side, we obtain $F_s = F + (A - A_s)$. Since $(A - A_s) \geq 0$, we have $F \leq F_s$.

(iii) By assumption, $A_s + F_s = \mathit{RCT}_i + \mathit{IBF}_i(A_s, A_s+F_s)$ and $\mathit{IBF}_i(A_s,\cdot) \equiv \mathit{IBF}_i(A,\cdot)$. Substitute $\mathit{IBF}_i(A_s,\cdot)$ in the former to obtain $A_s + F_s = \mathit{RCT}_i + \mathit{IBF}_i(A, A_s+F_s)$. Finally, we rewrite both sides using (i) to obtain $A+F = \mathit{RCT}_i + \mathit{IBF}_i(A, A+F)$. ◀

With Theorem 17 in place, we can obtain a result for any offset by (instead) analyzing an offset from the search space, which justifies the restriction to $A \in \mathcal{A}_i$ in Hypothesis 13.

4.3 Abstract RTA

In this section, the main proof is presented: using Theorems 15 and 17, we obtain that the constant R given in Hypothesis 13 is in fact a response-time bound for task τ_i.

The high-level idea of the proof is as follows. For an arbitrary job of task τ_i and its busy interval, we calculate the relative arrival time A. This offset is not necessarily in the search space, and therefore there is no direct access to the solution of the response-time recurrence with offset A. However, by Theorem 17, there is an equivalent solution of the recurrence for an offset included in the search space. We can use this solution to obtain an upper bound on interfering workload and use this bound to satisfy the premise of Theorem 15. In order to prove the theorem, we rely on all previously stated hypotheses.

▶ **Theorem 18** (Abstract RTA)**.** *Under the assumptions stated in Hypotheses 7, 8, 10, and 13, the response-time of task τ_i is bounded by R.*

We prove the theorem in the remainder of this section by case analysis in Lemmas 19–21. Let $J_{i,j}$ denote an arbitrary job of task τ_i. We must show that job $J_{i,j}$ is complete by time $a_{i,j} + R$ (or, equivalently, that $f_{i,j} \leq a_{i,j} + R$). By Hypothesis 7, there exists a finite busy interval $[t_1, t_2)$ of job $J_{i,j}$: let $A = a_{i,j} - t_1$ be the relative arrival time of job $J_{i,j}$ w.r.t. $[t_1, t_2)$. Further, as a zero-cost job trivially has a response time of zero, assume $c_{i,j} > 0$.

The goal is to apply Theorem 15 to obtain the time when $J_{i,j}$ receives sufficient service to become non-preemptive and run to completion, that is, we seek a Δ such that $\mathit{rct}_{i,j} \leq$

$RCT_i \leq serv_\sigma(J_{i,j}, [t_1, t_1 + \Delta))$. From Hypothesis 13, we can obtain the required bound on cumulative interference needed to satisfy the premise of Theorem 15 (as argued below).

Unfortunately, offset A does not necessarily belong to the search space; thus one cannot apply Hypothesis 13 directly. However, from Theorem 17 we know that, for any offset A, there exists another offset $A_s \in \mathcal{A}_i$ with an equivalent function IBF_i and solution F_s of the corresponding equation $A_s + F_s = RCT_i + IBF_i(A_s,\, A_s + F_s)$.

Depending on the value of $A_s + F_s$, consider the following cases: **(1)** $t_2 \leq t_1 + A_s + F_s$; **(2)** $t_1 + A_s + F_s < t_2$ and $A \leq A_s + F_s$; and **(3)** $t_1 + A_s + F_s < t_2$ and $A > A_s + F_s$. Clearly, these three cases cover all possibilities.

Case 1 In the first case, the solution $A_s + F_s$ of the response-time recurrence is larger than the length of the busy interval.

▶ **Lemma 19.** *If $t_2 \leq t_1 + A_s + F_s$, then $f_{i,j} \leq a_{i,j} + R$.*

Proof. We show that $f_{i,j} \leq t_2 \leq a_{i,j} + R$. The first inequality $f_{i,j} \leq t_2$ is a corollary of the definition of a busy interval (Definition 6). Time t_2 is a quiet time, and job $J_{i,j}$ arrives before t_2. Thus, by the definition of a quiet time (Definition 5), job $J_{i,j}$ is complete by time t_2. To prove the second inequality $t_2 \leq a_{i,j} + R$, consider the following chain of inequalities: $t_2 \leq t_1 + A_s + F_s \leq t_1 + A + F_s \leq a_{i,j} + F_s \leq a_{i,j} + F_s + (C_i - RCT_i) \leq a_{i,j} + R$. ◀

Case 2 Next, consider the case where the fixed point $A_s + F_s$ lies inside the busy interval, which is the theorem's main case.

Some additional reasoning is required since the term $C_i - RCT_i$ does not necessarily bound the term $c_{i,j} - rct_{i,j}$. That is, a job can have a small run-to-completion threshold $rct_{i,j}$, thereby becoming non-preemptive much *earlier* than guaranteed according to RCT_i, while simultaneously executing a final non-preemptive segment that is *longer* than RCT_i (*e.g.*, this is possible in the case of floating non-preemptive sections). In this case, we cannot directly apply Theorem 15, because the response-time recurrence gives a "weak" bound on workload with $\rho = RCT_i$, whereas we need $\rho = rct_{i,j}$ in this case.

Intuitively, however, this is a good situation: if a job has a longer final non-preemptive segment, it necessarily also has a shorter maximum preemptive part (since $c_{i,j} \leq C_i$). Observing that such a job will become non-preemptive earlier, the response time of the job will not be worse than the response time of a job with a shorter last segment. To formally express this reasoning, we introduce a notion of *optimism* $\mu \triangleq RCT_i - rct_{i,j}$.

▶ **Lemma 20.** *If $t_1 + A_s + F_s < t_2$ and $A \leq A_s + F_s$, then $f_{i,j} \leq a_{i,j} + R$.*

Proof. We show that $J_{i,j}$ will complete by time $a_{i,j} + R$ by deriving that it completes by time instant $t_1 + (A + F - \mu) + (c_{i,j} - rct_{i,j})$, where $t_1 + (A + F - \mu)$ is the point in time by which $J_{i,j}$ receives $rct_{i,j}$ units of service, and $c_{i,j} - rct_{i,j}$ is the duration of $J_{i,j}$'s last non-preemptive segment.

Due to space constraints, we omit the step-by-step proof of the inequality $t_1 + (A + F - \mu) + (c_{i,j} - rct_{i,j}) \leq a_{i,j} + R$, but note that we have verified this fact in our Coq proof.

We apply Theorem 15 with $\rho = rct_{i,j}$ and $\Delta = A + F - \mu$. It then remains to be shown that $rct_{i,j} + C_{\mathcal{I}}(J_{i,j}, [t_1,\, t_1 + A + F - \mu)) \leq A + F - \mu$, which trivially follows since $rct_{i,j} + C_{\mathcal{I}}(J_{i,j}, [t_1, t_1 + A + F - \mu)) \leq RCT_i + IBF_i(A, A + F) - \mu$. Thus we know that job $J_{i,j}$ receives at least $rct_{i,j}$ units of service by time $t_1 + A + F - \mu$, which implies that $J_{i,j}$ completes by time $t_1 + (A + F - \mu) + (c_{i,j} - rct_{i,j}) \leq a_{i,j} + R$. ◀

Case 3 The final case can never arise and thus can be discarded from further consideration.

▶ **Lemma 21.** *The case $t_1 + A_s + F_s < t_2$ and $A > A_s + F_s$ is impossible.*

Proof. By contradiction. Suppose the converse is true. From the equation $A_s + F_s = RCT_i + IBF_i(A_s, A_s + F_s)$, it follows that $rct_{i,j} + C_{\mathcal{I}}(J_{i,j}, [t_1, t_1 + A_s + F_s)) \leq A_s + F_s$. Thus, we can apply Theorem 15 with parameters $\rho = rct_{i,j}$ and $\Delta = A_s + F_s$. By Theorem 15, $J_{i,j}$ receives at least $rct_{i,j}$ units of service by time $t_1 + A_s + F_s$, where $0 < \varepsilon \leq rct_{i,j}$. However, this is impossible since $J_{i,j}$ arrives after time $t_1 + A_s + F_s$, and $J_{i,j}$ cannot receive service (*i.e.*, be scheduled) before it arrives. ◀

Lemmas 19–21 cover all cases, and in each possible case job $J_{i,j}$ completes by time $a_{i,j} + R$. Thus Theorem 18 holds.

5 Abstract Sequential Response-Time Analysis

In the analysis of uniprocessor systems, it is usually assumed that tasks are *sequential*, that is, jobs of the same task execute in order of their arrival. This assumption allows for tighter analyses because it allows for a better bound on *self-interference* (*i.e.*, the case when a job is, possibly indirectly, delayed by other jobs of the same task). In this section, we consider such an extension (*i.e.*, refinement) of the underlying abstract RTA, and to this end formally introduce the sequential-tasks assumption.

▶ **Hypothesis 22.** In schedule σ, *tasks are sequential*: for any task τ_i and any two jobs $J_{i,j}$ and $J_{i,k}$ of task τ_i, if $a_{i,j} < a_{i,k}$, then $\forall t,\ \sigma(t) = J_{i,k} \implies f_{i,j} \leq t$.

Assuming that tasks are sequential prevents a later-arriving job from interfering with an earlier job. This allows us to refine the interference bound function by isolating and removing the term that represents the self-interference contribution.

To support this property, in addition to Hypothesis 22, we require a technical condition to ensure that $\mathcal{I}_\sigma$ and $\mathcal{W}_\sigma$ are consistent with the sequential-tasks hypothesis (*i.e.*, to rule out nonsensical definitions of $\mathcal{I}_\sigma$ and $\mathcal{W}_\sigma$). Specifically, Hypothesis 22 can theoretically contradict the priority policy, which is implicitly encoded by the two functions, through the following effect. Suppose there are two jobs $J_{i,j}$ and $J_{i,k}$ of task τ_i that are pending simultaneously and $a_{i,j} < a_{i,k}$. By Hypothesis 22, $J_{i,j}$ must execute before $J_{i,k}$. However, since we have a completely generic, abstract model, nothing stops a (pathological) priority policy from assigning a higher priority to the second job (*i.e.*, $J_{i,k} \succ J_{i,j}$) so that $J_{i,k}$ must execute before $J_{i,j}$. To prevent such a contradiction, we impose an additional restriction on the interference function $\mathcal{I}_\sigma$ and the interfering workload function $\mathcal{W}_\sigma$.

▶ **Hypothesis 23.** *Functions $\mathcal{I}_\sigma$ and $\mathcal{W}_\sigma$ are consistent with Hypothesis 22*: for any job $J_{i,j}$ of task τ_i and its busy interval $[t_1, t_2)$, the total workload of the task up to time t_1 is equal to the total service received by the task up to time t_1. Formally, $wl^{\tau_i}([0, t_1)) = serv_\sigma^{\tau_i}([0, t_1))$.

To understand the intuition behind Hypothesis 23, recall that a busy interval is defined in terms of $\mathcal{I}_\sigma$ and $\mathcal{W}_\sigma$ (Definition 6). Thus, at the start of a job's busy interval, there should be no pending interference from other jobs of the same task. In other words, if upon arrival of a job $J_{i,j}$ there is already another pending job $J_{i,k}$ of the same task, then the busy interval of $J_{i,j}$ must extend to the arrival of job $J_{i,k}$.

Given Hypotheses 22 and 23, we can refine the response-time equation from Hypothesis 13 by pulling out the bound on self-interference from IBF_i. To this end and analogously to IBF_i, we let IBF_i^{other} denote a bound on any interference incurred by a job $J_{i,j}$ *excluding* any self-interference, and require that it satisfies the following invariant.

▶ **Hypothesis 24.** Given any job $J_{i,j}$ of a task τ_i and a schedule σ, $J_{i,j}$'s busy interval $[t_1, t_2)$ in σ, and a subinterval $[t_1, t_1 + \Delta) \subseteq [t_1, t_2)$, $\mathit{IBF}_i^{other}(a_{i,j} - t_1,\, \Delta)$ bounds the cumulative interference incurred by $J_{i,j}$ during $[t_1, t_1 + \Delta)$ *excluding* any self-interference: $C_{\mathcal{I}}(J_{i,j}, [t_1, t_1 + \Delta)) - (\mathit{serv}_\sigma^{\tau_i}([t_1, t_1 + \Delta)) - \mathit{serv}_\sigma(J_{i,j}, [t_1, t_1 + \Delta))) \le \mathit{IBF}_i^{other}(a_{i,j} - t_1,\, \Delta)$.

Using the new notion, we can refine IBF_i from Example 11 as follows.

▶ **Example 25.** An appropriate IBF_i^{other} function for a fully preemptive FP model with *sequential tasks* is given by

$$\mathit{IBF}_i^{other}(A, \Delta) \triangleq \sum \{ \mathit{RBF}_h(\Delta) \mid \forall \tau_h : \tau_h \neq \tau_i, \tau_h \succeq \tau_i \}\, .$$

Compared to IBF_i in Example 11, IBF_i^{other} explicitly excludes any contributions due to jobs of τ_i to discount any self-interference, which makes it less pessimistic for sequential tasks.

Recall from Section 2 that the system model assumes an arrival curve α_i to upper-bound the maximum number of activations of task τ_i, and that τ_i's request-bound function RBF_i is defined in terms of α_i. We use this information and Hypotheses 22–24 to upper-bound the maximum self-interference, and once we know the maximum self-interference, there is no need to include this term in the abstract function IBF_i^{other}. We state this fact as follows.

▶ **Lemma 26.** *Under Hypotheses 22–24, for any A and Δ:*

$$\mathit{IBF}_i(A,\, \Delta) \le \mathit{RBF}_i(A + \varepsilon) - C_i + \mathit{IBF}_i^{other}(A,\, \Delta).$$

Proof. Consider a job $J_{i,j}$ of task τ_i and its busy interval $[t_1, t_2)$, and let $A = a_{i,j} - t_1$ denote $J_{i,j}$'s relative offset. By Hypothesis 23, all jobs of task τ_i that arrived prior to time t_1 are complete by t_1. By Hypothesis 22, **(i)** all jobs of task τ_i that arrive during $[t_1, t_1 + A)$ *will* interfere with $J_{i,j}$, **(ii)** jobs that arrive *at* time $a_{i,j} = t_1 + A$ (simultaneously with $J_{i,j}$) *may* interfere with $J_{i,j}$, and **(iii)** *none* of the jobs of task τ_i that arrive after time $a_{i,j}$ can interfere with $J_{i,j}$. Therefore, we must consider only jobs of τ_i that arrive during the interval $[t_1, t_1 + A]$ of length $A + \varepsilon$. According to the system model (Section 2), these jobs execute for no more than $\mathit{RBF}_i(A + \varepsilon)$ time units in total. Furthermore, we can subtract τ_i's cost C_i once since $J_{i,j}$ itself is one of the jobs accounted for by $\mathit{RBF}_i(A + \varepsilon)$. The final inequality thus follows from Hypothesis 24. ◀

Lemma 26 implies that we can bound the term $\mathit{IBF}_i(A, A + F)$ in Hypothesis 13 with the refined term $\mathit{RBF}_i(A + \varepsilon) - C_i + \mathit{IBF}_i^{other}(A, A + F)$, which yields the following theorem.

▶ **Theorem 27.** *Under the assumptions stated in Hypotheses 7, 8, and 22–24, if there exists a constant R such that $\forall A \in \mathcal{A}_i, \exists F$,*

$$A + F = \mathit{RCT}_i + (\mathit{RBF}_i(A + \varepsilon) - C_i) + \mathit{IBF}_i^{other}(A,\, A + F) \text{ and}$$
$$F + (C_i - \mathit{RCT}_i) \le R,$$

then the response-time of task τ_i is bounded by R.

Proof. The claim follows from Theorem 18: the set of hypotheses remains mostly the same, with the only difference being that Hypotheses 10 and 13 have been refined with more specific assumptions. However, by Lemma 26, the refined bound $\mathit{RBF}_i(A{+}\varepsilon){-}C_i{+}\mathit{IBF}_i^{other}(A, A{+}F)$ safely upper-bounds $\mathit{IBF}_i(A, A + F)$; abstract RTA thus applies. ◀

With Theorem 27 in place, we next clarify when preemptions can take place.

6 Preemption Model

Recall from Section 2 that any job $J_{i,j}$ can be represented as a sequence of $q_{i,j}$ segments during which the job executes non-preemptively. Under a discrete-time model as assumed herein, this is true even if $J_{i,j}$ is fully preemptive since, as a degenerate case, such a job can be seen as consisting of $c_{i,j}$ single-quantum segments (*i.e.*, each of length ε).

Any non-preemptive segment starts with a corresponding *preemption point*, that is, a job can be preempted only in between segments. Accordingly, a *preemption model* is a policy that governs the placement of preemption points. Until now, we did not touch upon this aspect, because abstract RTA does not depend on any particular preemption model. Or rather, the specifics of the preemption model in use have been abstracted by the notion of RCT. However, to instantiate the analysis for specific schedulers and workloads, we need to specify an exact rule that determines when a job can be preempted.

To allow for a wide range of possible preemption models while retaining the ability to reason about preemptions in a high-level, general way, we use the notion of a *preemption predicate* $\psi : \mathbb{J} \times \mathbb{S} \to \mathbb{B}$ as a generic interface that abstracts from a job's specific structure and preemption points. More precisely, given any job $J_{i,j}$ and its *progress* $\rho \in [0, c_{i,j}]$ (*i.e.*, an amount of service received so far), predicate $\psi(J_{i,j}, \rho)$ holds iff $J_{i,j}$ can be preempted at this point of its execution (*i.e.*, after receiving exactly ρ units of service). We provide instantiations of ψ for concrete models in Appendix B. Interestingly, while one can think of a limited-preemptive model as a generalization (or super-model) of the other preemption models [7, 45], we found it actually easier to ignore this hierarchy in our proof. We thus instantiate ψ *directly* for each preemption model.

Since a preemption model restricts when a scheduler can enact changes to the schedule, we next relate the preemption predicate to a schedule σ and its underlying priority policy $\preceq$.

▶ **Definition 28.** *A schedule σ is* preemption-model compliant *if, for any job $J_{i,j}$ and time t, **(i)** $\psi(J_{i,j}, serv_\sigma(J_{i,j}, t)) \land \sigma(t) = J_{i,j}$ implies that $J_{i,j}$ has the maximal priority among all pending jobs and **(ii)** jobs can be non-preemptive only while they are executing: $\sigma(t) \neq J_{i,j} \implies \psi(J_{i,j}, serv_\sigma(J_{i,j}, t))$.*

In other words, all non-executing jobs are preemptable and the priority policy is respected whenever the scheduled job is preemptable, which in turn implies that jobs are preemptable at the moment of a context switch (*i.e.*, whenever they start or stop executing).

6.1 Priority Inversion

While a job $J_{l,k}$ is executing a non-preemptive segment, there may be a pending higher-priority job $J_{i,j}$ waiting to be scheduled at $J_{l,k}$'s next preemption point, which is commonly known as *priority inversion*. To reason about the total duration of priority inversion that $J_{i,j}$ incurs, we first define a weaker predicate $lps(J_{i,j}, t)$ that is true whenever a lower-priority job $J_{l,k}$ is scheduled, regardless of whether $J_{i,j}$ is actually pending at time t.

$$lps(J_{i,j}, t) \triangleq \exists J_{l,k},\ \sigma(t) = J_{l,k} \land J_{i,j} \succ J_{l,k} \tag{3}$$

Intuitively, it may seem somewhat unnatural to ignore whether $J_{i,j}$ is pending at time t. However, the definition turns out to be convenient because it captures cases where a priority inversion at the start of a busy interval indirectly influences the response times of jobs that arrive later in the busy interval. Since we consider priority inversion, and thus apply Equation (3), only within a busy interval, it is sufficient (and easier) to use $lps(J_{i,j}, t)$.

Table 1 Preemption Model Parameters

Preemption Model	RCT_i	B_i (where $\tau_l \in LP_i$)
Fully Preemptive	$C_i - \varepsilon$	$\varepsilon - \varepsilon = 0$
Fully Non-Preemptive	ε	$\max_l\{C_l - \varepsilon\}$
Fixed Preemption Points	$C_i - C_{i,m_i} + \varepsilon$	$\max_l\{\max_k\{C_{l,k}\} - \varepsilon\}$
Floating Non-Preemptive Regions	$C_i - \varepsilon$	$\max_l\{NPS_l - \varepsilon\}$

Therefore, building on Equation (3), we say that the *cumulative priority inversion* of a task τ_i is bounded by a constant B_i if, for any job $J_{i,j}$ of task τ_i and its busy interval $[t_1, t_2)$, it holds that $\sum_{t=t_1}^{t_2-1} [\![lps(J_{i,j}, t)]\!]_{\mathbb{1}} \leq B_i$.

While this notion of priority-inversion bound is more general than what is needed for non-preemptive segments, we chose it in anticipation of future work on locking protocols and other sources of priority inversion that can occur throughout a busy interval.

6.2 Bounded Non-Preemptive Segments

In this paper, we focus on priority inversions caused by lower-priority jobs executing non-preemptive segments of bounded length under a JLFP scheduling policy. To this end, we strengthen our assumptions about the schedule in the following.

▶ **Hypothesis 29.** The schedule σ is **(i)** preemption-model compliant (Definition 28) and **(ii)** *work-conserving in the classic sense*: for any job $J_{i,j}$ and any time t, if $J_{i,j}$ is pending and not scheduled at time t (*i.e.*, $a_{i,j} \leq t < f_{i,j}$ and $\sigma(t) \neq J_{i,j}$), then there exists another job that is scheduled at time t: $\sigma(t) \neq \bot$.

Classic work-conservation is a strong property that implies abstract work-conservation (when instantiated as in Section 7), and also the absence of self-suspensions, delayed budget replenishments, or any other rules or workload properties that defer a pending job's execution.

It follows from Hypothesis 29 that a busy interval contains at most one contiguous interval of priority inversion, which necessarily occurs at the beginning of the busy interval (if at all), and which is bounded by the maximum non-preemptive segment length.

The following theorem serves as an intermediate layer between abstract RTA, which has no explicit notions of preemption or priority inversion (both are considered abstract interference), and concrete schedulers and workload models, which come with specific preemption models. The benefit of this intermediate layer is that we can address several preemption models together, irrespective of which scheduling policies they are combined with, so that we do not need to repeat identical proofs as part of each instantiation.

▶ **Theorem 30.** *Under Hypothesis 29, the cumulative priority inversion of τ_i is bounded by $B_i \triangleq \max\{NPS_l - \varepsilon \mid \forall \tau_l \in LP_i\}$, where $LP_i \subseteq \tau$ denotes the subset of tasks that can generate jobs that can cause jobs of task τ_i to suffer priority inversion.*

We provide the proof in Appendix A. Intuitively, this theorem holds because NPS_l bounds the maximum non-preemptive segment length of any job of task τ_l, which means that for any job $J_{l,j}$ and at any point in its execution $\rho \in [0, c_{l,j}]$, there exists a $\delta \leq NPS_l$ such that $\psi(J_{l,j}, \rho + \delta)$ (*i.e.*, the job will be preempted no later than NPS_l time units after the beginning of a non-preemptive segment).

Based on Theorem 30, Table 1 states concrete instantiations of the four well-known preemption models for which we obtain concrete RTAs in the next section.

7 RTA Instantiation for FP and EDF

Finally, we put everything together to obtain concrete response-time bounds for EDF and FP via reductions to the sequential abstract model. Recall from Section 3 that we first must define the interface functions. As EDF and FP are both JLFP policies, we provide general definitions of $\mathcal{I}_\sigma$ and $\mathcal{W}_\sigma$ for a generic JLFP priority relation $\preceq : \mathbb{J} \times \mathbb{J} \to \mathbb{B}$.

Interference Conceptually, a job $J_{i,j}$ incurs interference whenever it is delayed (*i.e.*, prevented from being scheduled). Under a work-conserving JFLP policy, this occurs when either a higher- or equal-priority job preempts $J_{i,j}$, or when a lower-priority job causes priority inversion. In both cases, some job other than $J_{i,j}$ is scheduled, which results in the following trivial definition: $\mathcal{I}_\sigma(J_{i,j}, t) \triangleq \sigma(t) \neq J_{i,j}$.

Interfering Workload The definition of $\mathcal{W}_\sigma(J_{i,j}, t)$ is slightly more involved. Naturally, it includes the total cost of all higher- or equal-priority jobs released at time t. *Additionally*, $J_{i,j}$ accrues one time unit of interfering workload if there is a priority inversion at time t (*i.e.*, if $lps(J_{i,j}, t)$ holds) since no higher- or equal-priority workload is being "consumed" while a priority inversion persists.

$$\mathcal{W}_\sigma(J_{i,j}, t) \triangleq [\![lps(J_{i,j}, t)]\!]_{\mathbb{1}} + \sum \{ c_{h,k} \mid \forall J_{h,k} \in a(t) : J_{h,k} \succeq J_{i,j}, J_{i,j} \neq J_{h,k} \}$$

FP Interference Bound Next, we need to provide an appropriate bound on interference. It is well-known [3, 6, 29, 45] (and formally shown in our Coq proof) that

$$IBF_i^{other}(A, \Delta) \triangleq B_i + \sum \{ RBF_{hp}(\Delta) \mid \forall \tau_{hp} : \tau_{hp} \neq \tau_i : \tau_{hp} \succeq \tau_i \}$$

bounds interference under FP, with B_i given by Theorem 30 for $LP_i \triangleq \{ \tau_l \mid \tau_i \succ \tau_l \}$.

EDF Interference Bound It is further known [16] (and verified in our Coq proof) that

$$IBF_i^{other}(A, \Delta) \triangleq B_i + \sum \{ RBF_o(\min\{A + \varepsilon + D_i - D_o,\ \Delta\}) \mid \forall \tau_o : \tau_o \neq \tau_i \}$$

bounds interference under EDF, with B_i given by Theorem 30 for $LP_i \triangleq \{ \tau_l \mid D_o < D_i \}$.

To show that these reductions to the sequential abstract model are indeed correct, one needs to *prove* that Hypotheses 7, 8, and 22–24 are satisfied (which follows directly from properties of EDF, FP, and sporadic tasks). We omit these intuitively obvious proofs here for brevity, but stress that they form an integral part of the verified Coq proof.

As all hypotheses are satisfied, abstract sequential RTA (Theorem 27) yields the following verified general RTAs for *any* preemption model. By combining Theorems 31 and 32 below with the preemption models defined in Section 6, we obtain in total eight concrete, formally verified RTAs for FP and EDF.

▶ **Theorem 31.** *Given a preemption-model-compliant FP scheduler, a non-overloaded processor, and sequential, non-suspending tasks, if there exists an R such that $\forall A \in \mathcal{A}_i, \exists F$,*

$$\begin{aligned} A + F = B_i &+ (RBF_i(A + \varepsilon) - (C_i - RCT_i)) \\ &+ \sum \{ RBF_{hp}(A + F) \mid \forall \tau_{hp} : \tau_{hp} \neq \tau_i, \tau_{hp} \succeq \tau_i \} \end{aligned}$$

and $F + (C_i - RCT_i) \leq R$, where the search space is given by $\mathcal{A}_i \triangleq \{0\} \cup \{ A < L \mid RBF_i(A) \neq RBF_i(A + \varepsilon) \}$, then the response-time of task τ_i is bounded by R.

Interestingly, the definition of the search space $\mathcal{A}_i$ (*i.e.*, any offset A where RBF_i "steps") coincides with job releases in Lehoczky's critical-instant-based busy-window analysis [29]. The interference and self-interference bounds similarly match known results. We have thus independently obtained a proof of correctness of the well-known RTAs for FP, NP-FP, LP-FP, and FP-NPS scheduling [7, 45], notably without relying on a critical-instant argument. With the exception of FP scheduling [13], these bounds have not been formally verified before. In particular, Theorem 31 verifies Davis et al.'s revised CAN analysis [11]. For EDF, we obtain:

▶ **Theorem 32.** *Given a preemption-model-compliant EDF scheduler, a non-overloaded processor, and sequential, non-suspending tasks, if there exists an R such that $\forall A \in \mathcal{A}_i, \exists F$,*

$$\begin{aligned} A + F = B_i + &(RBF_i(A+\varepsilon) - (C_i - RCT_i)) \\ &+ \sum \{RBF_o(\min\{A+\varepsilon+D_i-D_o, A+F\}) \mid \forall \tau_o : \tau_o \neq \tau_i\} \end{aligned}$$

and $F + (C_i - RCT_i) \leq R$, where the search space is given by $\mathcal{A}_i \triangleq \{0\} \cup \{A < L \mid RBF_i(A) \neq RBF_i(A+\varepsilon) \ \vee\ \exists \tau_o \in \tau, \tau_o \neq \tau_i \ \wedge\ RBF_o(A + D_i - D_o) \neq RBF_o(A + \varepsilon + D_i - D_o)\}$, then the response-time of task τ_i is bounded by R.

While similar bounds were previously known for fully preemptive EDF [18, 20, 40], to the best of our knowledge, no RTAs have yet been proposed in prior work for NP-EDF, LP-EDF, or EDF-NPS. Further, no RTA for EDF has previously been verified, nor was it known that RTAs for EDF and FP share exactly the same proof scheme: namely, abstract RTA.

8 Conclusion and Future Work

Paul K. Harten Jr., in his 1987 paper [33] introducing one of the first RTAs for FP scheduling [28, 33],[3] which he obtained through formal reasoning in a temporal logic, stated as his closing remark: "The problem of developing a mechanical aid for the construction of proofs of real-time properties is a far more difficult one, and beyond the capabilities of this author" [33].

With the benefit of over 30 years of progress in the area of interactive theorem proving, by building on powerful tools and frameworks such as Coq [41] and Coq's Mathematical Components library [31], we have obtained the first fully mechanized formalization and machine-checked proof of the busy-window principle, a general proof scheme for mechanized RTAs that we call abstract RTA. Abstract RTA allowed us to include RTAs for eight practically relevant scheduler and workload combinations in Prosa [9, 34], which demonstrates its utility as a powerful foundation and proof framework for the development of verified RTAs.

Many opportunities for future work and further generalization remain. First, our definition of *RBF* currently precludes an efficient analysis of multi-frame task models, but this limitation can be easily removed. Second, it would be useful to introduce RTAs with support for precedence constraints. In a similar direction, it would be interesting to hoist Fradet et al.'s work [14] on a generalized digraph model in Prosa on top of our abstract RTA. Third, in this paper we focus only on the issue of proving that an instantiation of abstract RTA provides *sound* results, which is in keeping with Prosa project's focus on question of safety [9]. However, the problem of establishing necessary and sufficient conditions for an RTA to be precise is interesting and worth considering. Last but not least, abstract RTA is sufficiently general to obtain a verified RTA for self-suspending tasks, but obtaining a reasonably tight interference bound for such tasks (*i.e.*, instantiating IBF^{other}) remains a challenging problem.

[3] One of several independent, concurrent discoveries of the concept [2, 37].

References

1 AbstractRTA - ECRTS'20 Artifact Evaluation. URL: `https://people.mpi-sws.org/~sbozhko/ECRTS20/AbstractRTA.html`.

2 Neil C. Audsley, Alan Burns, Robert I. Davis, Ken Tindell, and Andy J. Wellings. Fixed priority pre-emptive scheduling: An historical perspective. *Real-Time Systems*, 8(2-3):173–198, 1995.

3 Neil C. Audsley, Alan Burns, Mike M. Richardson, Ken Tindell, and Andy J. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8(5):284–292, 1993.

4 Sanjoy K. Baruah. The limited-preemption uniprocessor scheduling of sporadic task systems. In *17th Euromicro Conference on Real-Time Systems, ECRTS 2005*, pages 137–144. IEEE Computer Society, 2005.

5 Marko Bertogna and Sanjoy K. Baruah. Limited preemption EDF scheduling of sporadic task systems. *IEEE Trans. Industrial Informatics*, 6(4):579–591, 2010.

6 Reinder J. Bril, Johan J. Lukkien, and Wim F. J. Verhaegh. Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption revisited. In *19th Euromicro Conference on Real-Time Systems, ECRTS 2007*, pages 269–279. IEEE Computer Society, 2007.

7 Reinder J. Bril, Johan J. Lukkien, and Wim F. J. Verhaegh. Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption. *Real-Time Systems*, 42(1-3):63–119, 2009.

8 John Carpenter, Shelby Funk, Philip Holman, Anand Srinivasan, James Anderson, and Sanjoy K. Baruah. A categorization of real-time multiprocessor scheduling problems and algorithms, 2004.

9 Felipe Cerqueira, Felix Stutz, and Björn B. Brandenburg. PROSA: A case for readable mechanized schedulability analysis. In *28th Euromicro Conference on Real-Time Systems, ECRTS 2016*, pages 273–284. IEEE Computer Society, 2016.

10 Jian-Jia Chen, Geoffrey Nelissen, Wen-Hung Huang, Maolin Yang, Björn B. Brandenburg, Konstantinos Bletsas, Cong Liu, Pascal Richard, Frédéric Ridouard, Neil C. Audsley, Raj Rajkumar, Dionisio de Niz, and Georg von der Brüggen. Many suspensions, many problems: a review of self-suspending tasks in real-time systems. *Real-Time Systems*, 55(1):144–207, 2019.

11 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.

12 Robert I. Davis, A. Zabos, and Alan Burns. Efficient exact schedulability tests for fixed priority real-time systems. *IEEE Trans. Computers*, 57(9):1261–1276, 2008.

13 Bruno Dutertre. Formal analysis of the priority ceiling protocol. In *21st IEEE Real-Time Systems Symposium, RTSS 2000*, pages 151–160. IEEE Computer Society, 2000.

14 Pascal Fradet, Xiaojie Guo, Jean-François Monin, and Sophie Quinton. A generalized digraph model for expressing dependencies. In *26th International Conference on Real-Time Networks and Systems, RTNS 2018*, pages 72–82. ACM, 2018.

15 Pascal Fradet, Xiaojie Guo, Jean-François Monin, and Sophie Quinton. CertiCAN: A tool for the Coq certification of CAN analysis results. In *25th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2019*, pages 182–191. IEEE, 2019.

16 Laurent George, Nicolas Rivierre, and Marco Spuri. Preemptive and Non-Preemptive Real-Time UniProcessor Scheduling. Technical Report RR-2966, INRIA, 1996.

17 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 2016*, pages 653–669. USENIX Association, 2016.

18 Nan Guan and Wang Yi. General and efficient response time analysis for EDF scheduling. In *Design, Automation & Test in Europe Conference & Exhibition, DATE 2014*, pages 1–6. European Design and Automation Association, 2014.

19 Xiaojie Guo, Maxime Lesourd, Mengqi Liu, Lionel Rieg, and Zhong Shao. Integrating formal schedulability analysis into a verified OS kernel. In *31st International Conference on Computer Aided Verification, CAV 2019*, pages 496–514. Springer, 2019.

20 José C. Palencia Gutiérrez and Michael González Harbour. Offset-based response time analysis of distributed systems scheduled under EDF. In *15th Euromicro Conference on Real-Time Systems, ECRTS 2003*, pages 3–12. IEEE Computer Society, 2003.

21 José C. Palencia Gutiérrez and Michael González Harbour. Response time analysis of EDF distributed real-time systems. *J. Embedded Computing*, 1(2):225–237, 2005.

22 Zain Alabedin Haj Hammadeh, Rolf Ernst, Sophie Quinton, Rafik Henia, and Laurent Rioux. Bounding deadline misses in weakly-hard real-time systems with task dependencies. In *Design, Automation & Test in Europe Conference & Exhibition, DATE 2017*, pages 584–589. IEEE, 2017.

23 Michael González Harbour and José C. Palencia Gutiérrez. Response time analysis for tasks scheduled under EDF within fixed priorities. In *24th IEEE Real-Time Systems Symposium, RTSS 2003*, pages 200–209. IEEE Computer Society, 2003.

24 Michael González Harbour, Mark H. Klein, and John P. Lehoczky. Fixed priority scheduling periodic tasks with varying execution priority. In *12th IEEE Real-Time Systems Symposium, RTSS 1991*, pages 116–128. IEEE Computer Society, 1991.

25 Rafik Henia, Arne Hamann, Marek Jersak, Razvan Racu, Kai Richter, and Rolf Ernst. System level performance analysis–the SymTA/S approach. *IEE Proceedings-Computers and Digital Techniques*, 152(2):148–166, 2005.

26 Kevin Jeffay, Donald F. Stanat, and Charles U. Martel. On non-preemptive scheduling of periodic and sporadic tasks. In *12th IEEE Real-Time Systems Symposium, RTSS 1991*, pages 129–139. IEEE Computer Society, 1991.

27 Mathai Joseph and Paritosh K. Pandya. Finding response times in a real-time system. *Comput. J.*, 29(5):390–395, 1986.

28 Paul K. Harter Jr. Response times in level structured systems. Technical Report CU-CS-269-84, University of Colorado, 1984.

29 John P. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *11th IEEE Real-Time Systems Symposium, RTSS 1990*, pages 201–209. IEEE Computer Society, 1990.

30 C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, 1973.

31 Assia Mahboubi and Enrico Tassi. Mathematical components, 2017.

32 Geoffrey Nelissen, José Carlos Fonseca, Gurulingesh Raravi, and Vincent Nélis. Timing analysis of fixed priority self-suspending sporadic tasks. In *27th Euromicro Conference on Real-Time Systems, ECRTS 2015*, pages 80–89. IEEE Computer Society, 2015.

33 Paul K. Harten Jr. Response times in level-structured systems. *ACM Trans. Comput. Syst.*, 5(3):232–248, 1987.

34 Prosa: The proven schedulability analysis repository, project web site. URL: `http://prosa.mpi-sws.org`.

35 Johannes Schlatow and Rolf Ernst. Response-time analysis for task chains in communicating threads. In *22th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2016*, pages 245–254. IEEE Computer Society, 2016.

36 Simon Schliecker, Jonas Rox, Matthias Ivers, and Rolf Ernst. Providing accurate event models for the analysis of heterogeneous multiprocessor systems. In *6th International Conference on Hardware/Software Codesign and System Synthesis, CODES+ISSS 2008*, pages 185–190. ACM, 2008.

37 Lui Sha, Tarek F. Abdelzaher, Karl-Erik Årzén, Anton Cervin, Theodore P. Baker, Alan Burns, Giorgio C. Buttazzo, Marco Caccamo, John P. Lehoczky, and Aloysius K. Mok. Real time scheduling theory: A historical perspective. *Real-Time Systems*, 28(2-3):101–155, 2004.

38 Lui Sha, Ragunathan Rajkumar, and John P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Trans. Computers*, 39(9):1175–1185, 1990.

39 Michael Short. Improved schedulability analysis of implicit deadline tasks under limited preemption EDF scheduling. In *16th IEEE Conference on Emerging Technologies & Factory Automation, ETFA 2011*, pages 1–8. IEEE, 2011.

40 Marco Spuri. Analysis of Deadline Scheduled Real-Time Systems. Technical Report RR-2772, INRIA, 1996.

41 The Coq proof assistant, project web site. URL: `https://coq.inria.fr`.

42 Ken Tindell, Alan Burns, and Andy J. Wellings. An extendible approach for analyzing fixed priority hard real-time tasks. *Real-Time Systems*, 6(2):133–151, 1994.

43 Yun Wang and Manas Saksena. Scheduling fixed-priority tasks with preemption threshold. In *6th International Workshop on Real-Time Computing and Applications Symposium, RTCSA 1999*, page 328. IEEE Computer Society, 1999.

44 Matthew Wilding. A machine-checked proof of the optimality of a real-time scheduling policy. In *10th International Conference on Computer Aided Verification, CAV 1998*, pages 369–378. Springer, 1998.

45 Gang Yao, Giorgio C. Buttazzo, and Marko Bertogna. Feasibility analysis under fixed priority scheduling with limited preemptions. *Real-Time Systems*, 47(3):198–223, 2011.

46 Man-Ki Yoon, Sibin Mohan, Chien-Ying Chen, and Lui Sha. TaskShuffler: A schedule randomization protocol for obfuscation against timing inference attacks in real-time systems. In *22th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2016*, pages 111–122. IEEE Computer Society, 2016.

47 Xingyuan Zhang, Christian Urban, and Chunhan Wu. Priority inheritance protocol proved correct. *J. Autom. Reasoning*, 64(1):73–95, 2020.

A Proof of Theorem 30

Recall that Equation (3) refers to a lower-priority job, thus a concrete priority-inversion bound B_i necessarily depends on the specific scheduling policy. However, the core of the argument is concerned only with the workload model. To separate the two concerns, we let $LP_i \subseteq \tau$ denote the subset of tasks that can generate jobs that (potentially) cause jobs of task τ_i to suffer priority inversion. The set LP_i serves to abstract from the specific priority policy; a concrete analysis instantiation must hence provide an appropriate definition of LP_i and prove its correctness. Based on these definitions, we next establish a priority-inversion bound for non-preemptive segments.

▶ **Theorem 30.** *Under Hypothesis 29, the cumulative priority inversion of τ_i is bounded by $B_i \triangleq \max\{NPS_l - \varepsilon \mid \tau_l \in LP_i\}$, where $LP_i \subseteq \tau$ denotes the subset of tasks that can generate jobs that (potentially) cause jobs of task τ_i to suffer priority inversion.*

Proof. Consider any job $J_{i,j}$ of task τ_i and its busy interval $[t_1, t_2)$. We need to show that $\sum_{t=t_1}^{t_2-1} [\![lps(J_{i,j}, t)]\!]_{\mathbb{1}} \leq B_i$. Suppose that $J_{i,j}$ incurs priority inversion at a time $t' \in [t_1, t_2)$. By Equation (3), some lower-priority job $J_{l,k}$ is scheduled at time t'. From clause (ii) of Hypothesis 29, it follows that, at any point during $J_{i,j}$'s busy interval $[t_1, t_2)$, either $J_{i,j}$ or a higher-priority job is pending. By Definition 28, this means that there is no preemption point at or prior to time t' (as otherwise $\sigma(t') \neq J_{l,k}$), and hence that $J_{l,k}$ is scheduled non-preemptively throughout (at least) $[t_1 - \varepsilon, t']$. As $J_{l,k}$ receives service throughout this interval, by the definition of NPS_l, there exists a preemption point at some time $t_1 - \varepsilon + \delta$ for $\delta \leq NPS_l$. Therefore, $t_1 \leq t' < t_1 - \varepsilon + NPS_l$ (*i.e.*, $t' \in [t_1, t_1 + NPS_l - \varepsilon)$). As we have made no assumptions on t', at most $NPS_l - \varepsilon$ points in $[t_1, t_2)$ satisfy $lps(J_{i,j}, t')$. The claim follows since $\tau_l \in LP_i$ by the definition of LP_i. ◀

B Concrete Preemption Models

Based on Theorem 30, we present concrete instantiations of four well-known preemption models, as summarized in Table 1. To define a preemption model, we must specify a suitable preemption predicate ψ and infer the corresponding task-level bounds NPS_i and RCT_i.

Fully Preemptive In a fully preemptive model, any job can be preempted at any point of its execution, and thus $\psi(J_{i,j}, \rho)$ is trivially true for any $\rho \in [0, c_{i,j}]$. From ψ, we can derive $NPS_i = \varepsilon$, and $RCT_i = C_i - \varepsilon$ (*i.e.*, a job executes "non-preemptively" and runs to completion only at the last moment before completion). From Theorem 30, we have $B_i = 0$.

Fully Non-Preemptive In a fully non-preemptive model, a job becomes non-preemptive as soon as it receives its first unit of service and then runs to completion, without any intermediate preemption points: $\psi(J_{i,j}, \rho) \triangleq \rho = 0 \vee \rho = c_{i,j}$. Thus $NPS_i \triangleq C_i$, $RCT_i \triangleq \varepsilon$, and $B_i = \max\{C_l - \varepsilon \mid \tau_l \in LP_i\}$.

Fixed Preemption Points For a model with fixed preemption points, also known as the *segmented limited-preemptive model*, each task τ_i is represented as a vector of m_i non-preemptive execution segments $C_{i,1}, C_{i,2}, \ldots, C_{i,m_i}$ such that $q_{i,j} = m_i$ and $0 \leq c_{i,j,k} \leq C_{i,k}$ for any job $J_{i,j}$ and all k. The corresponding preemption predicate is $\psi(J_{i,j}, \rho) \triangleq \exists x, \sum_{k=1}^{x} c_{i,j,k} = \rho$. Therefore, $NPS_i \triangleq \max\{C_{i,k} \mid \forall k \leq m_i\}$ and $RCT_i \triangleq C_i - C_{i,m_i} + \varepsilon$, which implies $B_i = \max_l\{\max_k\{C_{l,k}\} - \varepsilon\}$.

Floating Non-Preemptive Regions In a model with floating non-preemptive regions, jobs are usually (or mostly) preemptive, but contain non-preemptive segments of bounded length at *a priori* unknown points of their execution. In this model, NPS_i is a first-class task parameter (*i.e.*, it is provided as input, and not derived from other parameters). As in the case of fixed preemption points, the preemption predicate is $\psi(J_{i,j}, \rho) \triangleq \exists x, \sum_{k=1}^{x} c_{i,j,k} = \rho$, with the difference that $q_{i,j}$ is unknown *a priori*. That is, no assumption on the structure of each job is made; just that $c_{i,j,k} \leq NPS_i$ for all k. As the length of the last segment of a job can equal ε (*i.e.*, the job may be preemptive prior to completion), we have $RCT_i \triangleq C_i - \varepsilon$.