A Generalized Digraph Model for Expressing Dependencies

Pascal Fradet, Xiaojie Guo, Jean-François Monin, and Sophie Quinton

ABSTRACT
In the context of computer assisted verification of schedulability analyses, very expressive task models are useful to factorize the correctness proofs of as many analyses as possible. The digraph task model seems a good candidate due to its powerful expressivity. Alas, its ability to capture dependencies between arrival and execution times of jobs of different tasks is very limited.

We propose here a task model that generalizes the digraph model and its corresponding analysis for fixed-priority scheduling with limited preemption. A task may generate several types of jobs, each with its own worst-case execution time, priority, non-preemptable segments and maximum jitter. We present the correctness proof of the analysis in a way amenable to its formalization in the Coq proof assistant.

Our objective (still in progress) is to formally certify the analysis for that general model such that the correctness proof of a more specific (standard or novel) analysis boils down to specifying and proving its translation into our model. Furthermore, expressing many different analyses in a common framework paves the way for formal comparisons.

1 INTRODUCTION
The need for computer assisted verification of analysis techniques in the area of real-time systems has been recognized by the research community. The work presented here is part of an effort to contribute to Prosa [1], a library of definitions and proofs for real-time schedulability analyses using the Coq proof assistant [2]. Formal verification requires an important human effort, so making proofs general, generic, and/or reusable is of great importance.

Our goal is to factorize the formal certification of existing Response Time Analyses (RTAs) for fixed-priority policies. There exists a wide variety of task models and analyses for such policies. However, most of these models are incomparable and very few can describe dependencies between arrival and/or execution time of different jobs. The Digraph Real-Time Task (DRT) model [21] seems a good candidate for modeling intra-task dependencies, but its ability to capture dependencies between different tasks is very limited. It cannot, for example, capture Tindell’s offset model [25].

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

For the time being, the proof of the RTA of the general model within the Coq proof assistant is not yet complete. When it is certified, the design and proof of a more specific (standard or novel) RTA will boil down to specifying and proving its translation into our model. Furthermore, expressing many different RTAs in a common framework paves the way for formal comparisons and generalizations (e.g., design of novel RTAs).

To summarize, the main contributions of this paper are:

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

The paper is structured as follows. Section 2 introduces the systems we consider and provides basic definitions to describe their runtime behavior. Section 3 presents the syntax and semantics of our task model and Section 4 provides intuition about its expressivity. Section 5 presents the associated RTA and Section 6 sketches its correctness proof. Section 7 puts the contribution of this paper into the perspective of our broader project of a Coq library of schedulability results. We discuss related models in Section 8 and conclude in Section 9.

2 SYSTEM BEHAVIOR
We target concrete systems implemented as a set of tasks executing on a uniprocessor. The execution proceeds according to a job-level fixed-priority limited preemptive (JFPLP) scheduling policy. A JFPLP scheduler arbitrates between jobs competing for processor time by choosing the highest priority job, but it can only preempt a running job at some predefined execution point. In other words, each job is decomposed into non-preemptable segments. This model subsumes the Fixed Priority Preemptive (FPP) and Fixed Priority Non-Preemptive (FPNP) policies, and permits mixed policies [3]. A task can generate different types of jobs. Jobs of the same type have similar properties, in particular they have the same priority.
2.1 Definition of system behavior

We assume a set $\mathcal{T}$ of type names (types, for short). Each type entails a number of characteristics that are described in Section 2.

**Definition 1 (Jobs\textsuperscript{*}).** A job $j$ is specified by:
- its type $\tau(j) \in \mathcal{T}$;
- its priority $\rho(j) \in \mathbb{N}$ inherited from its type; A greater number means a higher priority.

Note that once the set of jobs and the function $\rho$ are given, $\rho$ is uniquely determined. A job $j$ completes when it has received as much service time as it required, which is determined by the schedule. We denote its completion time by $\tau(j) = a(j) + \hat{t}$.

A job $j$ is said to be pending.

**Definition 2 (Job arrival sequence\textsuperscript{*}).** A job arrival sequence is a function $\rho$ mapping any time instant $t$ to a finite (possibly empty) set of jobs $\rho(t)$ such that $j \in \rho(t)$ iff $a(j) = t$.

**Definition 3 (Schedule, JFPLP schedule\textsuperscript{*}).** A schedule is a partial function $\sigma$ which maps any time instant $t$ to the job (if any) that is scheduled (i.e., receives service) at $t$. A job $j$ can be scheduled only when it is pending.

A JFPLP schedule is a schedule such that the job that is scheduled is: either the job that is already executing one of its non-preemptable segments, or a job that has the highest priority $h$ among pending jobs. If there are several pending jobs with priority $h$, a task is arbitrarily selected among those with such jobs; the chosen job is then the first released job with priority $h$ in this task (FIFO policy).

2.2 Additional definitions and notations

In the following, we will need additional definitions and notations, which we introduce here.

**Definition 4 (Job release sequence).** Let $\rho$ be a job arrival sequence. The corresponding job release sequence, written $\hat{\rho}$, is defined as $\hat{\rho}(t) := \{ j \mid j \in \rho(t') \land t = t' + \hat{a}(j) \}$. We have $j \in \hat{\rho}(t)$ iff $\hat{r}(j) = t$.

We will occasionally write arrival (or release) sequences as lists of sets of jobs. For instance, $\{(t_1, \{j_1, j_2\}), (t_2, \{j_3\}), \ldots\}$ denotes a sequence $\rho$ such that $\rho(t_1) = \{j_1, j_2\}$, $\rho(t_2) = \{j_3\}$ and $\rho(t) = \emptyset$ for all $t$ absent from the list.

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

**Definition 5 (Workloads\textsuperscript{*}).** Let $\rho$ be a job arrival sequence and $V$ a set of job types. The workload $w_{\rho, V}$ of jobs with type in $V$ in a time interval $[t_1, t_1 + \Delta]$ is the cumulative cost (i.e., required service time) of such jobs released in that interval. Formally,

$$ w_{\rho, V}(t_1, \Delta) := \sum_{j \in V} c(j) t_{\leq j}(j) < t_1 + \Delta $$

**Definition 6 (Service times\textsuperscript{*}).** Let $\sigma$ be a schedule and $V$ a set of job types. The service times $w_{\rho, V}$ received by jobs with type in $V$ in a time interval $[t_1, t_1 + \Delta]$ is

$$ w_{\rho, V}(t_1, \Delta) := \sum_{\hat{r}(j) \in [t_1, t_1 + \Delta]} 1 $$

Our RTA analysis (Section 5) is based on the concept of busy window which we formally introduce now. The following definitions are implicitly parameterized by a job arrival sequence $\rho$ and a schedule $\sigma$.

**Definition 7 (Level-p quiet time\textsuperscript{*}).** An instant $t$ is said to be a level-p quiet time if all jobs of priority higher than or equal to $p$ released strictly before $t$ have completed at $t$.

**Definition 8 (Level-p busy window\textsuperscript{*}).** A time interval $[t_1, t_2]$ is said to be a level-p busy window if:

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

The last condition excludes degenerate cases of busy windows in which no job is scheduled. Since several jobs with the same type may be released in the same busy window, we introduce (again in a way similar to the state of the art) the additional concept of queueing prefix.

**Definition 9 (Queueing prefix\textsuperscript{*}).** The queueing prefix of jobs of type in a level-p busy window $[t_1, t_2]$ is the time interval $[t_1, t_q]$ where $t_q$ is the instant at which the last non-preemptable segment of the $q$-th job of type in the level-1 busy window (i.e., is scheduled for the first time).

**Example 1.** Figure 1 presents a job arrival sequence $\rho$ (split into $\rho_1$ and $\rho_2$) in a system made of two tasks. One task produces jobs of type $v$, priority 1, jitter 0 and segments (2, 2) and two consecutive arrival times of its jobs are separated by at least 9 time units. The other task produces jobs of priority 2. The three first queueing prefixes of jobs of type $v$ in the level-1 busy window $[0, 27]$ are $\mathcal{Q}_{\mathcal{Q}_1}(1, \mathcal{Q}_{\mathcal{Q}_2}(2) \mathcal{Q}_{\mathcal{Q}_3}(3)$.

3 GENERALIZED DIGRAPH TASK MODEL

Our task model, called the generalized digraph (GD) model, is an extension of the digraph model [21] with job level priorities, possibly null inter-arrival times, jitter and non-preemptable segments.
3.1 Syntax

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

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

A job type \( v \) is characterized by the following parameters:

- \( \rho(v) \in \mathbb{N} \) denotes the priority of jobs of type \( v \);
- \( J(v) \in \mathbb{N} \) specifies the maximum jitter (i.e., delay between arrival and release time) for jobs of type \( v \);
- \( C(v) = (C_1, \ldots, C_3) \) is a vector specifying the maximum cost of each non-preemptable segment of jobs of type \( v \); \( C(v) = \sum_{t=1}^{\Delta} C_t \) defines the maximum cost of jobs of type \( v \).

The sets of vertices \( V_i \) are assumed to be disjoint (i.e., tasks activate different job types). The set of vertices of the complete system is denoted by \( V_\Sigma = \bigcup_{i=1}^{n} V_i \). For simplicity and to improve readability, we assume in this paper that the jitter is constrained\(^3\): the jitter of any vertex \( v \in V_\Sigma \) is smaller than or equal to the minimum inter-arrival time labeled on any edge going out of \( v \).

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

In the following, we note \( h(p) \), \( h'(p) \), \( e(p) \), \( l(p) \) the sets of vertices of the system whose priorities are equal or higher, higher, equal and lower than \( p \), respectively.

**Example 2.** The graph \( G_e \) represented in Figure 2 defines a task with three vertices (i.e., three types of jobs). Vertex \( u \) is decorated with the triplet \( ((1, 1), 0, 1) \) where \( (1, 1) \) indicates that jobs of this type can be preempted at each instant (segments of length 1) and their maximum cost is \( 2 = 1 + 1 \); their maximum release jitter is 0 and their priority is 1. Similarly, jobs of type \( v \) have a maximum cost of 1, cannot be preempted, their maximum jitter and priority are 3 and 2, respectively. Jobs of type \( w \) have a maximum cost of 8 and can be preempted after at most 4 time units of execution.

3.2 Task-level and system paths

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

**Example 3.** In Figure 2, \([u, v, u, w, v], [w, v, u, v, w] \) and the infinite sequence \( X = [u, v, X] \) are paths of \( G_e \), but \([u, v, w] \) is not.

**Definition 10** (System path). A system path of system \( \Sigma \) is a set \( \pi := \{ \pi_1, \ldots, \pi_n \} \) such that for each \( i, \pi_i \) is a task-level path of task \( G_i \) in \( \Sigma \). The set of system paths of \( \Sigma \) is denoted \( \Pi_\Sigma \).

We will use later the following operations on task-level paths.

- The function \( \text{len} \) returns the sum of all minimum inter-arrival times on the edges of a finite path. Formally:
  \[
  \text{len}(v_1, v_2, \ldots, v_k) := \sum_{1 \leq j < k} d(v_j, v_{j+1})
  \]
  - The function \( \text{pre}_k^{\Pi}(\pi_i) \) returns the longest prefix \( \pi_p \) of \( \pi_i \) such that \( \text{len}(\pi_p) < \Delta \). A variant, written \( \text{pre}_0^{\Pi}(\pi_i) \), returns the prefix of \( \pi_i \) up to the \( n \)-th occurrence of vertex \( v \) in \( \pi_i \).
- We write \( \text{len}_n^{\Pi}(\pi_i) \) for \( \text{len}(\text{pre}_0^{\Pi}(\pi_i)) \) which returns the sum of all minimum inter-arrival times between the first vertex and the \( n \)-th occurrence of vertex \( v \) in \( \pi_i \).
- The function \( \text{cost}(\pi_i) \) returns the sum of the maximum cost of all vertices in a vertex sequence \( \pi_i \).
- The filter function \( \pi|_\pi \) returns the vertex sequence obtained from \( \pi \) where all vertices not belonging to \( V \) have been filtered out.

3.3 Semantics

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

**Definition 11** (Consistency of an arrival sequence w.r.t. a path). An arrival sequence \( \rho \) is \emph{consistent} with a task-level path \( \pi_i = [v_1, v_2, \ldots] \), which is denoted \( \rho \sim \pi_i \), iff there exists a flattening\(^4\) \( [\{t_1, j_1\}, \{t_2, j_2\}, \ldots] \) of \( \rho \) such that for all \( k \):

- \( J_k \) is consistent with \( v_k \), i.e.,
  - \( v(j_k) = v_k \);
  - \( p(j_k) = P(v) \);
  - \( f(j_k) \leq J(v_k) \); and

\(^3\)The extension to arbitrary jitter is discussed in Section 4.3.

\(^4\)For example, the arrival sequence \( [t_1, \{j_1, j_2\}, \{t_1, \{j_3\}\}] \) can be flattened into either \([\{t_1, j_1\}, \{t_1, j_2\}, \{t_1, j_3\}] \) or \([\{t_1, j_1\}, \{t_1, j_2\}, \{t_1, j_3\}] \).
and self-loop labeled with $\alpha$.

As a generalization of digraphs, the $G_d$ model naturally extends to system-level paths.

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

## 4 EXPRESSIVITY OF THE GD MODEL

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

### 4.1 Models without task dependencies

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

Arrival curves [24] represent more expressive arrival models. For instance, the minimal distance function $d(a)$ returns the smallest time interval that may contain $a + 1$ occurrences of a job. This function must be super-additive i.e., $d(a) + d(b) \leq d(a + b)$. In general, an arrival curve may have an infinite description in terms of $G_d$ task (e.g., $d(a) = a^2$ is super-additive and describes ever growing inter-arrival times). However, such functions[2] are usually given by a collection of values from 1 to some constant $k$. Then, the analysis uses the minimal super-additive extension (e.g., the smallest function compatible with $d(1), \ldots, d(k)$). Such super-additive closures can be represented faithfully by a $G_d$ task. When $d$ is convex (i.e., $d(x + 1) + d(x - 1) - 2d(x) \geq 0$) then the minimal super-additive extension is periodic and $\forall x \in k + r, d(x) = qd(k) + d(r)$. The corresponding $G_d$ task is then made of a cycle of $k$ vertices $v_0, \ldots, v_{k-1}$ where the minimum inter-arrival time decorating each edge is $d(v_i, v_{(i+1) \mod k}) = d((i + 1) \mod k) - d(i)$. $\Box$

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

For non convex functions, it has been shown that their minimal super-additive extensions are pseudo-periodic functions [10] which can also be represented by a finite $G_d$ task.

### 4.2 Models with job and task dependencies

As a generalization of digraphs, the $G_d$ model can of course express all task models with intra-task dependencies that standard digraphs can. Let us cite, the multiframe model [18] and its generalized version [6], the recurring branching [7] or recurring RT [8] models, the non cyclic RT [4] and digraph model (DRT) [21].

Allowing job-level fixed priorities and null minimal inter-arrival times allows the $G_d$ model to model inter-task dependencies

\[ d(v_k, v_{k+1}) \leq t_k + 1 - t_k \]

The definition naturally extends to system-level paths.

In Tindell’s model, a system is made of a set of independent transactions $\{T_{r1}, \ldots, T_{rn}\}$. Each transaction $T_{r_i}$ consists of a set of periodic tasks with offsets $T_{r_i} := \{v_{i, 0}, \ldots, v_{i, k}\}$ where each task $v_{i, k}$ has its own WCET, period, offset, priority, and deadline. A transaction and its set of periodic tasks with offsets can be represented within a single $G_d$ task. It is sufficient to compute the hyper-period of the transaction and to build the circular $G_d$ task representing the inter-arrival time (which may be null) between arrivals of the different jobs in the hyper-period. The periods and offsets are represented by inter-arrival times. The WCET, priority and deadline of tasks are represented by the corresponding vertices.

**Example 5.** Consider, for instance, a transaction $Tr$ with two periodic tasks with jobs of type $v_1$ and $v_2$, with periods 20 and 30 and with offsets 5 and 15. The hyper-period is 60 and taking the first arrival of $v_1$ as the time origin, the arrival times are $\{(t_1, 0), (v_2, 10), (v_1, 20), (v_1, 40), (v_2, 40), (v_1, 60)\}$. The transaction $Tr$ is represented by the $G_d$ task in Fig.3 (b). Its worst job arrival sequence w.r.t. the RTA is exactly the job arrival sequences of $Tr$. $\Box$

Shared resources, which entail inter-tasks dependencies, is a common issue in hard real-time systems. Abdullah et al. [3] addressed this problem using DRT by allowing two kinds of vertices: preemptable (tasks) and non-preemptable (resources). They proposed an extension of the RTA to take into account these two kinds of vertices. Using the $G_d$ model, this is directly modeled using segments with always preemptable vertices for tasks and non-preemptable ones for resources.

Rendez-vous mechanisms, another kind of inter-task dependencies, have been expressed using an extension of digraphs (SDRT) [17]. We believe that the encoding of such inter-task synchronization is possible in $G_d$ tasks but the exact encoding as well as its complexity remain to be investigated.

### 4.3 Beyond existing models

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

---

\footnote{By default $d(0) = 0$}

---

Figure 3: $G_d$ representing (a) an arrival curve model and (b) a transaction with offsets \textit{a la} Tindell (e.g., fixed timing relation among tasks), e.g., the offset model of Tindell [25] (which cannot be represented in the standard DRT model).

In Tindell’s model, a system is made of a set of independent transactions $\{T_{r1}, \ldots, T_{rn}\}$. Each transaction $T_{r_i}$ consists of a set of periodic tasks with offsets $T_{r_i} := \{v_{i, 0}, \ldots, v_{i, k}\}$ where each task $v_{i, k}$ has its own WCET, period, offset, priority, and deadline. A transaction and its set of periodic tasks with offsets can be represented within a single $G_d$ task. It is sufficient to compute the hyper-period of the transaction and to build the circular $G_d$ task representing the inter-arrival time (which may be null) between arrivals of the different jobs in the hyper-period. The periods and offsets are represented by inter-arrival times. The WCET, priority and deadline of tasks are represented by the corresponding vertices.

**Example 5.** Consider, for instance, a transaction $Tr$ with two periodic tasks with jobs of type $v_1$ and $v_2$, with periods 20 and 30 and with offsets 5 and 15. The hyper-period is 60 and taking the first arrival of $v_1$ as the time origin, the arrival times are $\{(t_1, 0), (v_2, 10), (v_1, 20), (v_1, 40), (v_2, 40), (v_1, 60)\}$. The transaction $Tr$ is represented by the $G_d$ task in Fig.3 (b). Its worst job arrival sequence w.r.t. the RTA is exactly the job arrival sequences of $Tr$. $\Box$

Shared resources, which entail inter-tasks dependencies, is a common issue in hard real-time systems. Abdullah et al. [3] addressed this problem using DRT by allowing two kinds of vertices: preemptable (tasks) and non-preemptable (resources). They proposed an extension of the RTA to take into account these two kinds of vertices. Using the $G_d$ model, this is directly modeled using segments with always preemptable vertices for tasks and non-preemptable ones for resources.

Rendez-vous mechanisms, another kind of inter-task dependencies, have been expressed using an extension of digraphs (SDRT) [17]. We believe that the encoding of such inter-task synchronization is possible in $G_d$ tasks but the exact encoding as well as its complexity remain to be investigated.

### 4.3 Beyond existing models

All these features, including jitter, can be combined to express and analyze new task models. For instance, non-preemptable segments and jitter have not been considered into intra-task dependencies task models (e.g., MF, GMF, RB, RR, DRT) nor do Tindell’s model consider intra-task dependencies (e.g., if a transaction could be a set of MF tasks instead of simple periodic tasks). Since all these models can be expressed as the $G_d$ model, they can be analyzed using the RTA described above.
The objective of RTA is to bound, as tightly as possible, the worst-case response time of each vertex (job type) \( v \), defined as the maximum response time among all jobs of type \( v \) occurring in all arrival sequences consistent with each possible system path. Formally,

\[
\text{wrt}(v) := \max \{ RT_j | j : v \land \exists \pi \in \Pi_{\Sigma}, \exists p \sim \pi, j \in \rho \}
\]

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

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

**Definition 12** (Busy window path). A level-\( p \) busy window \([t_1, t_2]\) is said to be represented by the shortest system path \( \pi \) such that \( \rho \approx [t_1, t_2] \sim \pi \), and any extension of it.

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

### 5.1 Overall structure of the RTA of Go systems

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

**Step 1:** Derive a set of system paths \( \Pi_{\Sigma}^v \) such that any possible level-\( p \) busy window (for any arrival sequence and schedule compatible with any system path in \( \Pi_{\Sigma} \)) is represented by one path in \( \Pi_{\Sigma}^v \).

**Step 2:** For each system path \( \pi \in \Pi_{\Sigma}^v \):

a) Compute an upper bound \( BW^*_{p, \pi} \) on the length of any level-\( p \) busy window represented by \( \pi \);

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

c) Compute, for each \( q \leq q^*_{v, \pi, BW^*_{p, \pi}} \), an upper bound \( Q_{\pi}^v(q) \) on the length of the \( q \)-th queuing prefix of jobs of type \( v \) in any level-\( p \) busy window represented by \( \pi \);

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

e) Based on the above, compute an upper bound \( RT_{\pi}^v(\pi) \) on the worst-case response time of any job of type \( v \) in any level-\( p \) busy window represented by \( \pi \).

**Step 3:** Finally, compute an upper bound \( RT^v(\pi) \) on \( \text{wrt}(v) \).

In the following subsection, we provide the formulas used for the computations associated with each step of the methodology that we just presented, along with some intuition of where they come from. The proof of correctness of the computed values is presented in Section 6.

### 5.2 Step-by-step RTA of Go systems

Our RTA relies on an upper bound on the workload of a set of jobs in any level-\( p \) busy window by the workload for the path in \( \Pi_{\Sigma}^v \) that represents it, where the workload of a given set of vertices \( V^q \subseteq V^v \) for a system-level path \( \pi := \{\pi_1, \ldots, \pi_n\} \in \Pi_{\Sigma}^v \) and a duration \( \Delta \) is

\[
\text{wt}^v_{V, \pi}(\Delta) := \sum_{s=1}^n \text{wt}^v_{V, \pi_s}(\Delta)
\]

with

\[
\text{wt}^v_{V, \pi_s}(\Delta) := \text{cost}(\text{pre}_{A^v} + J^v_{\pi_s})(\pi_s) \| v
\]

We will show (see Lemma 2 in Sec. 6) that the workload of a set of vertices \( V^q \subseteq V^v \) for any prefix of length \( \Delta \) of a level-\( p \) busy window represented by a system path \( \pi \) is upper bounded by \( \text{wt}^v_{V, \pi}(\Delta) \).

**Step 1: Computing \( \Pi_{\Sigma}^v \) ** — see Theorem 1 in Sec. 6

Let \( \Pi_{\Sigma}^v(\Delta) \) denote the set of paths \( \pi_x \) of task \( G_x \in \Sigma \) which:

- are the longest paths fitting in \( \Delta + J^v_{\pi} \), that is, such that \( \text{len}(\pi_x) \leq \Delta + J^v_{\pi} \) and for all valid suffixes \( s, \text{len}(\pi_x.s) > \Delta + J^v_{\pi} \); where \( J^v_{\pi} \) representing the largest release jitter among \( V^v \);
- if the vertex under study \( v \) belongs to \( G_i \) then we consider only paths \( \pi_x \) with occurrences of \( v \).

With this notion of path, we compute a sufficiently large duration which can bound the length of any level-\( p \) busy window. That duration is the least positive fixed point, written \( W_k \), of the following equation

\[
\Delta = N + \sum_{x=1}^n \max_{\pi_x \in \Pi_{\Sigma}(\Delta)} \left\{ \text{wt}^v_{\text{rep}(p_s), \pi_x}(\Delta) \right\}
\]

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

\[
\Pi_{\Sigma}^v := \prod_{x=1}^n \Pi_{\Sigma}(W_k)
\]
where $\bigtimes_{n=1}^{\infty}$ denotes the Cartesian product of all paths of length bounded by $W_k$ for the $n$ tasks. Thus, any level-$p$ busy window can be represented by (possibly a prefix of) a path in $\Pi^\Sigma_w$.

**Step 2: Computing upper bounds for each path in $\Pi^\Sigma_w$**

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

a) **Computing $BW^+_{p,\pi}$ — see Theorem 2 in Sec. 6.**

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

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

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

$$B_p := \max_{v \in C(lp(p))} (C - 1)$$

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

$$\Delta = B_p + \max_{v \in C(lp(p))} \left( \min_{\pi_i} len^\Sigma_{\pi_i}(\pi_i) - J(fst(\pi_i)) \right)$$

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

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

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

Definition 12 implies that the number of jobs of type $v$ in a busy window is equal to the number of $v$ in its representing path. In addition, since $BW^+_{p,\pi}$ is an upper bound on the length of any possible level-$p$ busy window represented by $\pi$, let $q^+_{v,\pi_i,BW^+_{p,\pi}}$ be the number of vertices $v$ in the prefix $pre_{BW^+_{p,\pi}} + j(fst(\pi_i))(\pi_i)$ of path $\pi_i$, then $q^+_{v,\pi_i,BW^+_{p,\pi}}$ is an upper bound on the number of jobs of type $v$ released in any level-$p$ busy window represented by $\pi$.

c) **Computing $Q^+_{v,\pi}(q)$ — see Theorem 3 in Sec. 6.**

The $q$-th queuing prefix of jobs of type $v$ in a level-$p$ busy window represented by $\pi$ spans the execution of:

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

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

$$\Delta = B_p + \max_{v \in C(lp(p))} \left( \min_{\pi_i} len^\Sigma_{\pi_i}(\pi_i) - J(fst(\pi_i)) \right)$$

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

d) **Computing $\theta^+_{v,\pi_i}(q)$ — see Theorem 4 in Sec. 6.**

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

$$\theta^+_{v,\pi_i}(q) := len^\Sigma_{\pi_i}(\pi_i) - J(fst(\pi_i))$$

e) **Computing $RT^+_{\pi}(v)$ — see Theorem 5 in Sec. 6.**

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

$$RT^+_{\pi}(v) := \max_{q \leq q^+_{v,\pi_i,BW^+_{p,\pi}}} \left( Q^+_{v,\pi}(q) + \theta^+_{v,\pi_i}(q) + last(\tilde{C}(v)) - 1 \right)$$

**Step 3: Computing $RT^+_{\pi}(v)$**

Finally, performing the same computation for all paths in $\Pi^\Sigma_w$ yields the worst-case response time of jobs of type $v$.

$$RT^+_{\pi}(v) := \max_{\pi \in \Pi^\Sigma_w}(RT^+_{\pi}(v))$$

Note that the max functions in Equation 10 and 11 play different roles: The first one (Eq. 10) accounts for the fact that there may be several jobs of the same type (vertex) in a level-$p$ busy window; the second one (Eq. 11) allows a fine-grained analysis of dependencies (which are captured by the notion of path).

**5.3 Improvements**

For the sake of clarity, we have presented the analysis by first computing a superset of possible paths and then focusing on a single vertex. Clearly, this approach is not the most efficient. First, the paths considered are larger than needed and the analysis is likely to consider many time the same prefix common to many paths. Second, as presented, a complete system analysis would need to iterate the same process for each vertex. Both points entail costly and/or useless recomputations.

A more reasonable approach is to analyse pertinent paths only once. An analysis of the whole system would proceed by considering all possible alignments between vertices of all tasks. For each alignment $\{v_1, \ldots, v_q\}$, we compute the level-$p$ busy window (with $p$ the minimal priority) for all possible paths starting from the considered alignment. Paths are built on demand; they end when a level-$p$ quiet time is reached. That busy window is the longest and includes all other busy windows. The computation should keep enough information to evaluate $Q^+_{v,\pi}(q)$, $\theta^+_{v,\pi_i}(q)$ and therefore $RT^+_{\pi}(v)$ for each vertex $v$ in the path. Finally, the maximum of
We outline here the proofs of the main lemmas used to establish 
$q$\(\hat{\text{window}}\) over all alignments gives the worst-case response time for any vertex $v$ (i.e., jobs of type $v$).

Further improvements can be made. Consider a vertex $v_i$ in an alignment \{v$_1$, ..., v$_n$\}, then if there is some $v_j$ with a lower priority than $v_i$ whereas task $G_j$ has another vertex with a priority equal or higher than $v_i$, then this alignment cannot be a worst case for $v_i$ and $RT^*_H(v_i)$ does not need to be computed in that alignment.

The analysis described so far is precise; the only source of approximation lies in the blocking factor $B_p$ (see Eq. 6). However, depending on the size of $G_0$, considering all possible alignments may be overly expensive. Approximations should be studied. A possible approximation consists in computing a single over-approximated workload function for each priority and each task. This lower drastically the number of combinations to test (more on this in the Sec. 7). Note that this approach is similar to the approximation used by Tindell in his offset analysis [26] and by Guan et al. in their DRT analysis [15]. In fact, since our approach generalizes these two analyses, existing techniques should be applicable.

6 PROOF OF CORRECTNESS

We outline here the proofs of the main lemmas used to establish the correctness of the RTA presented in Sec. 5. Our short term goal is to complete these proofs using Coq and the Prosa library (see Section 7). In contrast with classical proofs in the RT community, machine-verified proofs require to list all used hypotheses, to specify formally concrete executions and to prove many properties usually taken for granted.

In the following, we consider a JFPLP schedule $\sigma$ and a system-level job arrival sequence $\rho \sim \Sigma$ having a job $j$ of type $v$ with priority $p$ and belonging to task $G_i$. Any such job occurs within a level-$p$ busy window. Therefore, we consider that $j$ occurs as the $q$-th job of type $v$ in a level-$p$ busy window starting at instant $t_1$.

6.1 Concrete busy window and queueing prefix

The key to the proof of the RTA is to show the correctness of the upper/lower bounds (e.g., $BW^+_{p, \pi}$) computed using the abstract model (i.e., $G_0$). So, we need to specify formally the length $BW_p$ of the considered level-$p$ busy window and the length $Q_{v, p}(q)$ of the $q$-th queueing prefix of jobs of type $v$ in order to bound them.

Length of the level-$p$ busy window

The length of a level-$p$ busy window starting at $t_1$, denoted by $BW_p$, can be computed as the least positive fixed point of the following equation.

$$\Delta = serv_{\rho(p), \sigma}(t_1, \Delta) + hep(\rho_p)(\rho(p), t_1, \Delta)$$

(12)

The first term represents the service provided to the possible non preemptable segment of a lower priority job. Then, the amount of services performed by the scheduler for $hep(\rho)$ jobs is equal to the workload requested from $hep(\rho)$ within the duration of the busy window.

Length of the $q$-th queueing prefix

Similarly, computing the length of the $q$-th queueing prefix of jobs of type $v$ (i.e., the queueing prefix of $j$) in the level-$p$ busy window $[t_1, t_1 + BW_p]$ amounts to evaluate:

- the service provided to the possible non preemptable segment of a lower priority job;
- the workload of jobs of task $G_i$ with the priority $p$ (minus the cost of $j$’s last segment);
- the workload of jobs of vertices from $G_i$ in $hep(\rho)$;
- the workload of other jobs with priorities in $hep(\rho)$.

Thus, the length of the $q$-th queueing prefix of jobs of type $v$, denoted by $Q_{v, p}(q)$, can be defined as the least positive fixed point of the following equation.

$$\Delta = serv_{\rho(p), \sigma}(t_1, \Delta)
+ hep(\rho_p)(\rho(p), t_1, \Delta)
+ hep(\rho_p)(\rho(p), t_1, \Delta)$$

(13)

6.2 Basic lemmas

To bound $BW_p$ and $Q_{v, p}(q)$ we rely on the following lemma.

Lemma 1 (x). Let $f, g : \mathbb{N} \rightarrow \mathbb{N}$ be two functions and $\Delta_1$ and $\Delta_2$ be fixed points of the equations $\Delta = f(\Delta)$ and $\Delta = g(\Delta)$ then, if for all $x : \mathbb{N}$, $f(x) \leq g(x)$ and, for all $x : \mathbb{N}^+$, $x < \Delta_1$, we have $x < f(x)$, then $\Delta_1 \leq \Delta_2$.

Proof. Easy proof by contradiction. \hfill $\square$

Using this lemma the proof that $BW^+_{p, \pi}$ and $Q^+_{v, p}(q)$ upper bound $BW_p$ and $Q_{v, p}(q)$ respectively, amounts to show that $serv$ and $hep$ are increasing and to compare the $rhs$ of their recursive definitions.

A key step to this aim is to bound the workload of a set of jobs of types in $V$ during a busy window by the workload of the set of vertices in $V$ of the path representing that busy window. More generally, we prove the following lemma:

Lemma 2. Let $\Delta$ be a time interval and let $\pi := \{\pi_1, \ldots, \pi_n\}$ be the system path representing $\hat{\rho}_{|t_1, t_1 + \Delta}$ i.e., $\hat{\rho}_{|t_1, t_1 + \Delta} \sim \pi$, then

$$wl_{V, \pi}(t_1, \Delta) \leq \sum_{x=1}^{n}wl_{V, \pi_x}(t_1, \Delta)$$

(14)

Furthermore, that bound is tight.

Proof. The arrival sequence $\rho$ can be decomposed into $n$ independent task-level job arrival sequences $\{\rho_1, \ldots, \rho_n\}$ where each $\rho_x$ is the arrival sequence of the jobs of task $G_x$. Then,

$$wl_{V, \pi}(t_1, \Delta) = \sum_{x=1}^{n}wl_{V, \pi_x}(t_1, \Delta)$$

Recall the definition of

$$wl_{V, \pi}(\Delta) := \sum_{x=1}^{n}cost(\{pre_{\Delta + 1}\}f_{\pi_x}(\pi_x)_{\pi_x})$$

Therefore, it is sufficient to prove, for all task $G_x$ that

$$wl_{V, \pi_x}(t_1, \Delta) \leq \sum_{x=1}^{n}cost(\{pre_{\Delta + 1}\}f_{\pi_x}(\pi_x)_{\pi_x})$$

We show that the workload of $\hat{\rho}_{x/|t_1, t_1 + \Delta|$ is maximal when:

1. any two consecutive jobs of $\rho_x$ are separated by their minimum inter-arrival time $d$;
2. all jobs take their maximum cost (i.e., WCET) $C$.
(3) the first job in the interval releases at $t_1$ after having experienced its maximum release jitter (if $fist(\pi_x))$ whereas the jitter of all other jobs is null.

When these three conditions are met, the value of $w_{hep(p),\pi_x}(t_1, \Lambda)$ is maximal and exactly equal to $cost(\{pre_{\Lambda+\succeq f(j\succeq \pi_x)}(\pi_x)\} + \) That directly implies Lemma 2.

6.3 Correctness of $\Pi^U_{\Sigma}$
As a first step towards the correctness proof of the RTA, we must show that any possible level-$p$ busy window is represented by one path in $\Pi^U_{\Sigma}$.

Theorem 1. Any level-$p$ busy window is represented by at least one path in $\Pi^U_{\Sigma}$ as defined in Equation 5.

Proof. It suffices to show that $W_k$ bounds the length of any level-$p$ busy window. i.e.,

$$BW_p \leq W_k$$

Recall that $W_k$ is the least positive fixed point of the equation

$$\Delta = N + \sum_{x=1}^{n} \max_{\pi_x \in \Pi^U_{\Sigma}(\Lambda)} \{w_{hep(p),\pi_x}(\Lambda)\}$$

We first prove that the rhs of Equation 16 bounds the one of Equation 17. It is fairly easy to prove that $B_p \leq N$ since $N$ represents the largest segment in the system. Furthermore, for each task $G_k \in \Sigma$, we clearly have

$$w_{hep(p),\pi}(\Lambda) \leq \max_{\pi_x \in \Pi^U_{\Sigma}(\Lambda)} \{w_{hep(p),\pi_x}(\Lambda)\}$$

and therefore, for the system path and all tasks,

$$w_{hep(p),\pi}(\Lambda) \leq \sum_{x=1}^{n} \max_{\pi_x \in \Pi^U_{\Sigma}(\Lambda)} \{w_{hep(p),\pi_x}(\Lambda)\}$$

Then, Lemma 1 entails $BW^+_{p,\pi} \leq W_k$ and Theorem 2 permits to establish Equation 5 by transitivity.

6.4 Correctness of bounds for job $j$
Now, we show the correctness of the upper/lower bounds $BW^+_{p,\pi}$, $q^+_{v,\pi,\sigma}$, $Q^+_{v,\pi}(q)$ and $\tilde{C}(\pi)$ which implies the correctness of the upper-bound $R^+_{\pi}(\pi)$ for the response time of $j$. Let $\pi$ be the path representing the level-$p$ busy window $\{t_1, t_1 + BW_p\}$.

Theorem 2. Let $BW^+_{p,\pi}$ be the least positive fixed point of the equation

$$\Delta = B_p + w_{hep(p),\pi}(\Lambda)$$

then

$$BW_p \leq BW^+_{p,\pi}$$

Proof. We first prove that the rhs of Equation 17 bounds the rhs of

$$\Delta = serv_{\{p\},\sigma}(t_1, BW_p) + w_{hep(p),\rho}(t_1, \Lambda)$$

The definition of a level-$p$ busy window implies that at most one non-preemptable segment of any vertex in $\Pi(p)$ of $\pi$ can execute in $\{t_1, t_1 + BW_p\}$. Further, such a segment (if it exists) must have started execution before $t_1$. Therefore, $serv_{\{p\},\sigma}(t_1, BW_p)$ is bounded by $B_p$. Lemma 2 ensures that the second term of the rhs of equation 19 is bounded by $w_{hep(p),\pi}(\Lambda)$. Then, Lemma 1 permits to conclude.

Lemma 3. The number of vertices $v$ in $\pi_1$ upper-bounds $q$;

$$q \leq q^+_{v,\pi,\sigma,\rho}(p)$$

Proof. This result follows by the definition of a path representing a busy window and $q^+_{v,\pi,\sigma,\rho}(p)$.

Theorem 3. Let $Q^+_{\pi}(q)$ be the least positive fixed point of the equation

$$\Delta = B_p + w_{hep(p),\pi}(\Lambda)$$

and then it bounds the length of the $q$-th queueing prefix of jobs of type $\pi$ in the level-$p$ busy window i.e.,

$$Q^+_{\pi}(q) \leq Q^+_{\pi}(q)$$

Proof. As for Theorem 2, it suffices to prove that the rhs of Equation 20 bounds the rhs of Equation 13. We prove it by considering each term in turn.

(1) In Theorem 2, we proved $serv_{\{p\},\sigma}(t_1, BW_p) \leq B_p$. Further, the definition of a queueing prefix implies that $Q^+_{\pi}(q) \leq BW_p$. Therefore, for all $\Delta \leq BW_p$, (which is sufficient for Lemma 1), $serv_{\{p\},\sigma}(t_1, \Lambda) \leq BW_p$. Let $\{t_1, BW_p\}$.

(2) The second term can be divided in two parts:

$$w_{hep(p),\pi}(\Lambda) \leq \sum_{x=1}^{n} \max_{\pi_x \in \Pi^U_{\Sigma}(\Lambda)} \{w_{hep(p),\pi_x}(\Lambda)\}$$

Then, Lemma 1 entails $BW^+_{p,\pi} \leq W_k$ and Theorem 2 permits to establish Equation 5 by transitivity.

Now, it is sufficient to prove that (25) $\leq (23)$ and (26) $\leq (24)$.

- According to Definition 12, the type of any job of $hep(p) \cap V_i$ released in $\{t_1, \alpha(j) + 1\}$ occurs $pre_\pi(\alpha(i)) + \alpha(i-1)$. Taking into account the minimum inter-arrival time, we have

$$w_{hep(p),\pi}(\Lambda) \leq \sum_{x=1}^{n} \max_{\pi_x \in \Pi^U_{\Sigma}(\Lambda)} \{w_{hep(p),\pi_x}(\Lambda)\}$$

If we filter out $j$ from $\pi$ and the $q$-th $v$ (job $j$’s type) from $\pi$, (25) $\leq (23)$ follows.
- By definition, we know that $\tilde{C}(\pi) = \langle c_1, \ldots, c_s \rangle$, $\tilde{C}(v) = \langle C_1, \ldots, C_s \rangle$ and, for all $i = 1, \ldots, s, c_i \leq C_i$. So

$$\sum_{i=1}^{s-1} C_i \leq \sum_{i=1}^{s-1} C_i$$

Equivalently $\tilde{C}(j) \leq \tilde{C}(v)$. Therefore (26) $\leq (24)$ follows.
The last two inequalities between the last two terms of (21) and of (13) follow directly from Lemma 2.

Theorem 4. The term $\theta_{v,n_i}^-(q) = \text{len}_v^q(\pi_i) - J(fst(\pi_i))$ is a lower bound of the duration between the arrival of $j$ and the beginning of its level-$p$ busy window.

$$\theta_{v,n_i}^-(q) \leq a(j) - t_1$$ (27)

Proof. By cases.

1. $a(j) - t_1 < 0$. The job $j$ arrives before $t_1$ but releases at or after $t_1$ i.e., $t_1 \leq a(j) + j(j)$. Constraining jitter implies that no other job of the same task arrives before the release of $j$. So job $j$ must be the first vertex of $\pi_i$ and $\text{len}_v^q(\pi_i) = 0$. By convention, $j(j) \leq J(v)$, therefore $-J(v) \leq -j(j) \leq a(j) - t_1$ and equation 27 holds.

2. $a(j) - t_1 \geq 0$. Let $j'$ be the job corresponding to the first vertex in $\pi_i$. We know $a(j) - a(j') \geq \text{len}_v^q(\pi_i)$ and $j(j') \leq J(fst(\pi_i))$. Then, $\text{len}_v^q(\pi_i) - J(fst(\pi_i)) \leq a(j) - a(j') - j(j')$ and since $a(j') + j(j') \geq t_1$ that is $-a(j') - j(j') \leq t_1$, equation 27 follows.

Lemma 4. Let $RT_j$ be the response time of job $j$ then

$$RT_j \leq Q_{v,p}^+(q) - \theta_{v,n_i}^-(q) + \text{last}(\tilde{C}(v)) - 1$$ (28)

Proof. By definition $RT_j := \text{end}(j) - a(j)$. Also, when a job begins to execute its last non-preemptable segment it cannot be preempted until its completion. Using the notion of $q$-th queuing prefix, we have

$$\text{end}(j) = t_1 + Q_{v,p}^+(q) + \text{last}(\tilde{C}(j)) - 1$$

Note that $Q_{v,p}^+(q)$ includes the first cost unit of job $j$’s last segment, so we should subtract 1 from $\text{last}(\tilde{C}(j))$. Equivalently, we have

$$RT_j = Q_{v,p}^+(q) - a(j) - t_1 + \text{last}(\tilde{C}(j)) - 1$$

The result follows from Theorem 3, Theorem 4 and the fact that costs in the model upper-bounds costs in the execution.

6.5 Correctness of the RTA

The previous result applies to an arbitrary job $j$ of type $v$ in a busy window starting at $t_1$. It can be used to upper-bound the response times of any job of type $v$ in that busy window, and by extension any job of type $v$ in the arrival sequence.

Theorem 5. The response time of any job of type $v$ released in a busy window starting at $t_1$ is bounded by

$$\max_{q \leq q_{v,n_i}, BW_{p,v}^+} \left\{ Q_{v,p}^+(q) - \theta_{v,n_i}^-(q) + \text{last}(\tilde{C}(v)) - 1 \right\}$$ (29)

Proof. Follows from properties of max and Lemma 4.

Theorem 6. The response time of any job of type $v$ released in any arrival sequence $\rho \sim \Sigma$ is bounded by

$$\max_{p \in H_\Sigma^P} \max_{q \leq q_{v,n_i}, BW_{p,v}^+} \left\{ Q_{v,p}^+(q) - \theta_{v,n_i}^-(q) + \text{last}(\tilde{C}(v)) - 1 \right\}$$ (30)

Proof. Follows from properties of max and Theorem 5.

7 DISCUSSION

In this section, we discuss the significance of the presented model and analysis in the context of our broader effort toward a Coq library of schedulability results.

In the past decades, there have been a few attempts at proving methods [13, 14] for solving real-time problems. Recently, the Prosa library [11, 12] has been proposed to provide formal specifications and mechanized proofs for schedulability analyses using the Coq proof assistant. The motivation behind our general task model for fixed priority scheduling is to add it to the Prosa library and prove the correctness of its RTA. It can thus cover a large variety of existing models and analyses.

7.1 Proving in Coq the RTA of Go systems

The complete Coq proof of the RTA for Go systems is still in progress and can be separated into two parts:

1. The generic proof of RTAs for the JFLP scheduling policy. For this part, many definitions (i.e., those in Sec. 2) have been formalized and used in Prosa, as well as a significant part of the proof. We are actually formalizing the proof of a more general statement, which does not rely on a task model, but on an abstract workload function

$$w_{fr} : (\rho \rightarrow N \rightarrow N \rightarrow T) \rightarrow N \rightarrow N$$

where: (a) the first argument ($\rho \rightarrow N \rightarrow N \rightarrow T$) denotes a function taking a job arrival sequence, a time instant and a time duration and returning an abstract candidate represented by the type $T$, where candidates correspond to incomparable scenarios which must be analyzed, e.g., paths for the Go model; (b) the second argument denotes a time duration such that $w_{fr}$ returns the workload during that duration. The proof as well as the analysis are then applicable to many task models respecting the fixed priority scheduling policy, including the Go model, by instantiating that function.

2. Specifying the Go task model and instantiating the function $w_{fr}^*$. This part is not formalized yet; however, according to our experience, it should not raise any issue.

7.2 Intended use of the analysis

One of our objectives is to formally certify our RTA in order to:

- compare it with existing RTAs for task models which can be expressed by Go, in terms of precision and time complexity;
- obtain novel machine-verified RTAs which take into account e.g., jitter, non-preemptable segments and offsets;
- reuse the generic part of the proof to propose machine-verified RTAs for task models beyond Go by focusing on upper bounding the workload.

7.3 Beyond the current analysis

By proposing a unified analysis for models as different as the DRT model and Tindell’s offset model, our work underlines the generic parts of the proof structure of such RTAs. Based on this, we can now propose a framework which formalizes these steps in a generic manner, to be reused for any new task model. Such steps include
the use of sustainability properties [5, 11], but also strategies to efficiently approximate the worst-case response time.

8 RELATED WORK

Many task models have been proposed to analyze different fixed-priority scheduling policies. Depending on their capacity to model intra- or inter-task dependencies, those models can be divided into three categories.

The simplest models do not consider any kind of dependency: there is only one type of job for each task. The classic periodic task model, presented by Liu and Layland [16] characterizes a task by its worst-case execution time \( C \) and its activation period \( P \). The sporadic task model [19] generalizes activation periods of tasks by introducing the concept of minimum inter-arrival time. Later Thiele et al. introduced the real-time calculus [24], whose arrival curves can model many more arrival patterns. Another category of models considers intra-task dependencies: there may be several types of jobs for each task. The multiframe model [18], characterizes a task by an array of execution times \((C_0, C_1, \ldots, C^{N-1})\) and a minimum inter-arrival time \( P \). A task has \( N \) types of jobs and the \((i + 1)\)-th job in the arrival sequence has the worst-case execution time \( C_{(i \mod N)} \) and arrives at least \( P \) time units after the arrival time of the \( i \)-th job. The model \((C, P, D)\) extends the multiframe model by allowing each type of job to have a different inter-arrival time and deadline [6]. Later, Baruah introduced the recurring branching task model [7], which adds branching structures. Each task can be represented by a tree of job types \((C, D)\) labeled by minimum inter-arrival times. Two other extensions use directed acyclic graphs instead of trees: the recurring real-time task model [8, 9], and the non-cyclic recurring real-time task model [4]. More recently, Stigge et al. introduced the DRT task model [21] and its extended version [22] that use arbitrary graphs. None of those models allows to model inter-task dependencies.

One of the most classic task model taking inter-task dependencies into account is Tindell’s offsets model. A system is made of a collection of transactions regrouping periodic tasks having fixed timing relations. More recently, the DRT model was extended by Mohaqeqi et al. to allow the RDV mechanism [17] and by Abdullah et al. to take into account shared resources [3].

To the best of our knowledge, none of the previous models is general enough to express at the same time intra- and inter-task dependencies as well as arrival curves. Our model is a generalization in this respect. The formal certification of its associated RTA should permit to factorize the correctness proofs of many analyses.

9 CONCLUSION

In this paper, we have introduced the G\( o \) model, a generalization of the DRT task model that is expressive enough to model and analyze many different fixed-priority systems. In particular, G\( o \) can express dependencies between jobs as well as tasks. The work presented in this paper is motivated by our ongoing contribution to Prosa, a Coq library of models and analyses of real-time systems. The G\( o \) model and its associated RTA provide the needed foundations for a Coq response time analysis of complex systems, in particular regarding dependencies. Future work includes:

- The complete Coq formalization of the presented RTA, which is still in progress.
- A formal comparison of our proposed analysis with the existing RTAs of specific types of DRTs, e.g., constrained deadline under job-level FPP or task-level FPNP [23], and arbitrary deadline under task-level FPP [20]. Indeed, our RTA for G\( o \) uses a queuing prefix technique which may require a smaller number of paths to be analyzed.
- A practical study of the complexity of the analysis and of the possible trade-off between accuracy of the computed bounds and runtime performance of an RTA implementation.
- Extensions to more complex models, in particular to task chains and multiprocessor systems.
- A theoretical connection between the RTA proposed here and the notion of sustainability.

We believe that this work represents a significant step toward integrating previously independent features into a unified framework for the response time analysis of real-time systems.

ACKNOWLEDGMENT

This work has been partially supported by the LabEx PERSYVAL-Lab (grant ANR-11-LABX-0025-01) through the CASERM project and the French national research organization ANR (grants ANR-15-CE25-0008 and ANR-17-CE25-0016) through the VOCAL and RT-PROOFS projects. We would like to thank Maxime Lesourd for insightful discussions.

REFERENCES


[19] A. K.-L. Mok. Fundamental design problems of distributed systems for the hard-


[25] K. Tindell. Using offset information to analyse static priority pre-emptively sched-