A Formal Link Between Response Time Analysis and Network Calculus

Pierre Roux
ONERA, Toulouse, France
DTIS – Université de Toulouse, F-31055 Toulouse, France

Sophie Quinton
Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG, F-38000 Grenoble, France

Marc Boyer
ONERA, Toulouse, France
DTIS – Université de Toulouse, F-31055 Toulouse, France

Abstract

Classical Response Time Analysis (RTA) and Network Calculus (NC) are two major formalisms used for the verification of real-time properties. We offer mathematical links between these two different theories. Based on these links, we then prove the equivalence of various key notions in both frameworks. This enables specialists of both formalisms to get increase confidence on their models, or even, like the authors, to discover errors in theorems by investigating apparent discrepancies between some notions expected to be equivalent. The presented mathematical results are all mechanically checked with the interactive theorem prover Coq, building on existing formalizations of RTA and NC. Establishing such a link between NC and RTA paves the way for improved real-time analyses obtained by combining both theories to enjoy their respective strengths (e.g., multicore analyses for RTA or clock drifts for NC).

2012 ACM Subject Classification Computer systems organization → Real-time system specification; Networks → Formal specifications; Software and its engineering → Formal methods; General and reference → Verification

Keywords and phrases Response Time Analysis, Network Calculus, dense time, discrete time, response time, formal proof, Coq

Digital Object Identifier 10.4230/LIPIcs.ECRTS.2022.5

Supplementary Material *Software (ECRTS 2022 Artifact Evaluation approved artifact)*: `https://doi.org/10.4230/DARTS.8.1.3`

Funding This work has been partially supported by the French national research organization ANR (grant ANR-17-CE25-0016) through the RT-PROOFS project.

1 Introduction

Classical Response Time Analysis (RTA) and Network Calculus (NC, together with its variant Real-Time Calculus) are two major formalisms used for the verification of real-time properties. Some of the differences between RTA and NC are well-known: RTA tends to be based on discrete time while NC relies on dense time, there is no notion of task in NC, etc. Still, fully understanding the implications of such differences – enough, for example, to be able to compare the state of the art in both approaches – requires a solid expertise in both formalisms, which very few people have. To the best of our knowledge, no formal link has ever been proposed to relate models and verification techniques from both worlds. This is now made easier by recent work on formalizing each technique separately using the Coq interactive theorem prover: RTA in Prosa [9] and NC in NCCoq [22].

34th Euromicro Conference on Real-Time Systems (ECRTS 2022).
Editor: Martina Maggio; Article No. 5; pp. 5:1–5:22
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In this paper, we provide the foundations for a formal connection between RTA and NC. Specifically, we show how to translate the behavior of a real-time system as formalized in Prosa into a trace as specified in NCCoq, and conversely. This requires in particular a clean formalization of time and how to switch back and forth between discrete and dense time. We also relate core modeling concepts of NC, namely arrival curves and FIFO scheduling, to their Prosa counterparts. All definitions and proofs are formalized in Coq and available as additional material submitted together with this paper.

Our work is significant in many ways. First, it makes it easier for experts of one of the two approaches to understand the other by formally relating definitions. Second, given that a formal specification in Coq may be incorrect (meaning that it does not correspond to its informal definition), linking RTA and NC definitions is a way to increase confidence in these definitions. This has in fact led us to discover a bug in the definition of static priority in NC, as discussed at the end of the paper. Third, our formal connection provides the necessary foundations to compare existing analyses: proving that they are equivalent, that one is strictly more precise than another, or that they are incomparable. In addition, we hope that this connection can be used to build improved analyses based on a combined use of RTA and NC. Last, but not least, another strong contribution of this paper is its formalization of clocks. Discrete time is not a well suited setting for addressing issues related to clock drifts. The formal connection provided here between discrete and dense time represents a first step towards handling clock drifts in Coq.

This paper is organized as follows. Section 2 discusses related work. Section 3 provides an informal overview of the contribution. Sections 4 and 5 then present in a formal way the relevant state of the art regarding modeling in Coq of RTA and NC, respectively. Section 6 presents our first contribution, which formally links physical time (as in NC) and discrete clock-based time (as in Prosa). Sections 7, 8 and 9 then provide formal links between arrival sequences and cumulative curves, response times and horizontal deviations, request bound functions and arrival curves, respectively. Finally, Section 10 addresses properties at the scheduling level regarding FIFO and fixed-priority scheduling and Section 11 concludes.

All along the paper, definitions and lemmas are tagged with their (`name`) in the companion Coq development. No pen-and-paper proof of any result is provided in the paper: we focus instead on intuitions and explanations. Of course, we can only do that because the provided Coq proofs provide a much higher level of confidence.

2 Related Work

Building an analysis to guarantee that a system satisfies some real-time requirement is often a complex process, requiring long and error-prone proofs. One way to increase confidence in the correctness of such analyses is to use model checking, as e.g., in [5] to verify schedulers. Model checkers are automatic but less versatile than proof assistants such as Coq [10] or Isabelle/HOL [26]. In this paper, we follow a recent trend in the real-time community towards computer-assisted formal specifications and proofs using Coq[1].

Coq [10] is a proof assistant, i.e., a tool offering (1) a language to state theorems and describe their proofs, and (2) software – think of it as a compiler – for verifying these proofs. It can also be used to develop software whose execution is proved to be conform to their (formal) specification such as the CompCert C compiler [19] or the CertiKOS operating

[1] See for example the Call to Action at ECRTS 2016, *Real Proofs for Real Time: Let's do better than "almost right"* [1]. Note that a similar momentum was given a decade ago in the programming language community. A number of mechanized formalizations (using either Coq or other tools) now appear each year at POPL, their main conference.

system [13]. When checking a proof, Coq will complain if a lemma is used without providing a proof for one of its hypotheses or if the proved hypotheses do not match the expected ones. Of course, Coq cannot guarantee that the formalization indeed corresponds to what was intended, so readability of the specification is key in Coq.

The main mathematical concepts of RTA have been formally defined and proved, with Coq, in the Prosa library [9]. This work has since seen many extensions. For example, [8] uses Coq as much for formalization purposes as for verification, while [7, 12] both use Coq as a powerful tool for providing abstract proofs which can then be instantiated into a variety of more concrete analyses. CertiCAN [11] represents a third type of application of Coq in relation to Prosa, in that it not only produces Coq proofs of various real-time analyses of the CAN protocol (from which efficient analysis tools can be extracted) but it can also be used to certify the results of non-certified industrial tools such as RTaW-Pegase. In yet another related line of work, [14, 15] have shown that it is possible to use Prosa for the schedulability analysis of a real-time OS kernel, namely CertiKOS.

On the NC side, first results on the formal verification of NC computation were presented in [20]. The aim was to verify that an existing tool was correctly using the NC theory. An Isabelle/HOL library was developed, specifying the main concepts of NC (flows and servers, arrival and service curves) and stating the main theorems but without proving them[2]. The NC tool was then instrumented to provide not only a result, but also a proof on how NC had been used to produce that result. Isabelle/HOL was in charge of checking the correctness of this proof. Much more recently, actual proofs, this time using Coq, of the core mathematical concepts of NC have been provided in [22]. Actual computations of such non trivial manipulations of functions in the min-plus dioids can also be verified using Coq [23].

Regarding the specific contribution of this paper, namely the link between RTA and NC, note that comparing theories that coexist in the real-time community is not a new challenge [21], yet little research has been done on building bridges between them. Recent work has focused on connecting Compositional Performance Analysis (CPA, [24]) and NC into a common formal framework [6, 17], however not using proof assistants.

3 Overview of the Contribution

Before going into detail, let us provide here an informal description of the contributions of this paper, based on simple examples of RTA and NC execution traces.

Figure 1 shows a usual representation of a RTA trace, which contains here a single *job* characterized by its *arrival time* $arr(j)$ and its *cost*, both of which are integers. The scheduling of j determines the service it receives, and thus its *completion time* $end(j)$. Note that cost and instants are all integers but instants denote some points in time whereas a cost is a workload. The response time of j is defined as $end(j) - arr(j)$. A more detailed description of RTA will be presented in Section 4.

Figure 2 shows a comparable trace in NC. NC models workload using the notion of *cumulative curve*. A cumulative curve A is a non-decreasing function from $\mathbb{R}_+$ to $\mathbb{R}_+$, whose semantics is that $A(t)$ represents a cumulative amount of work (in bits or CPU cycles). Like in RTA, the domain and image of A are the same sets, but the domain represents a (dense) time set whereas the image represents an amount of work. A curve can represent an amount of work demand, or an amount of work received. For instance, the crossing of a server by a packet is represented by two functions, the arrival cumulative curve A, whose value is null up to the packet arrival and is increased by the packet size at its arrival time, and a

[2] They were assumed to be correct, since they have been established in the literature for long.

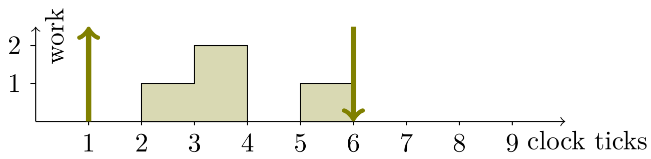

Figure 1 RTA: Scheduling of a job j with $arr(j) = 1$, $cost(j) = 4$ and the service received by job equal to 1 at instants 2 and 5, and equal to 2 at instant 3, leading to $end(j) = 6$.

departure cumulative curve D which is also null up to the packet departure and is increased by the packet size at its departure time. The delay is then defined as the horizontal deviation between A and D (the formal definition of which will be given in Section 5).

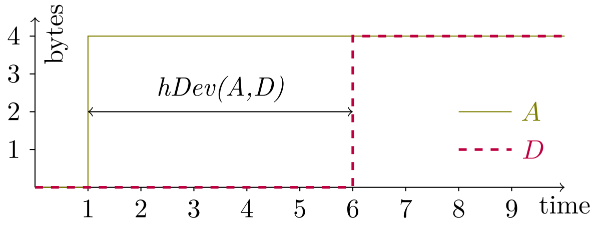

Figure 2 NC: Arrival and departure cumulative curves representing a packet of size 4 entering a server at time 1 and leaving at time 6.

The relation between RTA and NC appears quite simple when comparing Figures 1 and 2, but the similarity between the graphical horizontal lines hides a major difference. Time in NC is the physical time, common to all calculators or switches (neglecting relativity effects) whereas time in RTA is the clock time, given by a hardware element, subject to imprecision and drifts.

Let us now discuss a slightly more complex example. In RTA, a *task* is a (possibly infinite) sequence of jobs, and the response time of the task in a schedule is the maximum of all individual response times. In NC, a *flow* is commonly represented by a single cumulative sequence. Note that the information is not exactly the same in both models. First, the NC model does not precisely represent the instants at which a task is scheduled, and only represents completion times. Since it does not represent the scheduling itself, it cannot support properties on scheduling. Second, the NC model of flow has no notion of individual packets or jobs. For example, when looking at Figure 3, representing the arrival and departure of two packets of size 4, one cannot guess if the packet leaving at time 6 is the first or the second one. In fact, even the number of packets is unknown: the jump of size 4 at time 1 could be created by a single packet of size 4, but also by 2 packets of size 2, and so on. Nevertheless, under the assumption that all jobs of the same task are scheduled with FIFO order, we can prove that the horizontal deviation is equivalent to the response time. Third, the RTA model does not capture clock drifts, making it more difficult to plug systems with different clocks, whereas NC uses the real universal time and can easily handle clock drift between systems.

We can now detail the core contributions reported in this paper, as follows:

- a clock model linking RTA and NC time, that can handle clock drifts;
- a mapping of each job j in a Prosa trace to a pair (A_j, D_j) in NC;
- a proof that the horizontal deviation $hDev(A_j, D_j)$ is equal to the response time of job j in case of perfect clocks, and a valid upper bound in presence of clock drifts;

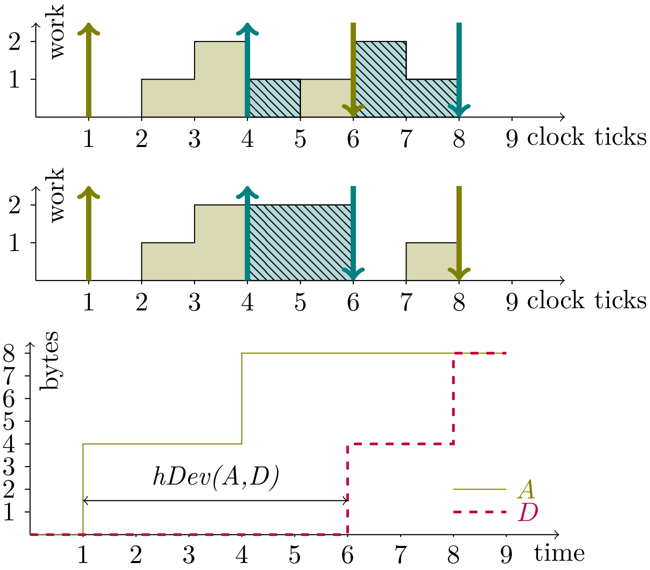

Figure 3 Graphical representation of two different schedulings of two jobs j, j' and the associated arrival and departure curves (A, D). Note that NC curves cannot distinguish the two schedulings.

- similar results for entire tasks and not just single jobs;
- a translation of a request bound function from RTA into an arrival curve from NC and vice versa;
- an equivalence between RTA and NC definitions of FIFO;
- an error found in a NC theorem during some preliminary works toward an equivalence on static priority.

All these items have been formalized in Coq based on the Prosa and NCCoq frameworks.

As already mentioned, this equivalence is only based on job release time, completion time and cost, ignoring the scheduling itself. Then, the property on FIFO relies on the observable part related to input/output, not on its internal implementation. And the result on static priority than will be presented in Section 10.2 is only a preliminary work. The modeling of the schedule will be discussed in the conclusion.

4 Response Time Analysis

Let us provide a more formal description of how the general concepts used by RTA are specified in Prosa. The Prosa library is structured around three main parts:

- `behavior/` provides a trace-based semantics of real-time systems and is meant to be as generic as possible. This part represents the common ground for all other parts of Prosa.
- `model/` contains a variety of modeling concepts which can be used to specify real-time systems, e.g., regarding task arrivals, preemptions, scheduling policies, platform abstractions etc. This part is meant to be used as a library, where one can pick definitions suitable for a specific system model.
- `analysis/` is where response time or schedulability analysis proofs are located.

In the rest of this section, we present the definitions from the Prosa library that are needed to relate RTA and NC. For readability, we omit trivial well-formedness conditions and a few basic definitions.

4.1 Behavior

The `behavior` part of Prosa is the core of the library, so most of its concepts are used in our work. First, let us recall that Prosa is based on a discrete model of time, so time is defined using natural numbers. This is formalized in Coq in the file `time.v` of Prosa as

```
Definition duration := nat.
Definition instant  := nat.
```

To insist on this and avoid confusion with the dense time used by NC, we use in this paper the term *tick* for what is called instant in Prosa, and *number of ticks* for durations.

A *job* in Prosa is an abstract type with decidable equality: given two jobs, one can at least determine whether they are the same job or not. One can specify additional job parameters such as cost or arrival. Note that the cost of a job, which denotes the amount of service it requires to complete, is of type `work` (also represented using natural numbers), which would correspond in a real system to a number of processor cycles.

▶ **Definition 1.** (`job_cost`, `job_arrival`) *The cost of a job j, of type `work`, is denoted $\mathit{cost}(j)$. The arrival of a job j, of type `tick`, is denoted $\mathit{arr}(j)$.*

This is formalized in Coq in the file `job.v` of Prosa as

```
Definition JobType := eqType.
Definition work  := nat.
Class JobCost (Job : JobType) := job_cost : Job -> work.
Class JobArrival (Job : JobType) := job_arrival : Job -> instant.
```

An arrival sequence is then defined as a function mapping any tick to the (finite) sequence of jobs that arrive at that tick.

▶ **Definition 2.** (`arrival_sequence`) *Given a type of jobs Job, an arrival sequence is a function $\mathit{arrseq} : \mathbb{N} \to 2^{\mathit{Job}}$.*

This is formalized in Coq in the file `arrival_sequence.v` of Prosa as

```
Definition arrival_sequence (Job : JobType) := instant -> seq Job.
```

A schedule essentially specifies which jobs are scheduled at every tick, and how much service they receive. In practice, depending on the specific execution platform, a lot more information may be relevant. This is why Prosa provides an abstract notion of processor state, which provides at least the above information about jobs scheduled and service. A schedule is then defined as a function that maps a tick to a processor state.

▶ **Definition 3.** (`schedule`) *A schedule is a function $\mathit{sched} : \mathbb{N} \to \mathit{PState}$. Given an instance of the abstract processor state class, the service received by a job j in a processor state $p \in \mathit{PState}$ is denoted $\mathit{service\_in}(j, p)$.*

This is formalized in Coq in the file `schedule.v` of Prosa as

```
Definition schedule (PState : Type) := instant -> PState.
```

with

```
Class ProcessorState (Job : JobType) (PState : Type) :=
  { service_in : Job -> PState -> work; (* ... *) }.
```

Given a schedule and an instance of the abstract notion of processor state, one can derive the service received by a job at a given tick, between two given ticks, or up to a given tick.

▶ **Definition 4.** (`service`) *Given a schedule sched, the service received by a job j before a tick $t \in \mathbb{N}$, denoted $service(j, t)$, is defined as $\sum_{0 \leq t' < t} service\_in(j, sched(t'))$.*

This is formalized in Coq in the file `service.v` as

```
Context {Job : JobType} {PState : Type} `{ProcessorState Job PState}.
Variable sched : schedule PState.

Definition service_at (j : Job) (t : instant) := service_in j (sched t).
Definition service_during (j : Job) (t1 t2 : instant) :=
  \sum_(t1 <= t < t2) service_at j t.
Definition service (j : Job) (t : instant) := service_during j 0 t.
```

Finally, a job completes its execution when it has received at least as much service as its cost[3]. The definition of a response time bound follows.

▶ **Definition 5.** (`job_response_time_bound`) *A number of ticks r^+ is a response time bound for a job j if $service(j, arr(j) + r^+) \geq cost(j)$.*

This is formalized in Coq as

```
Definition completed_by (j : Job) (t : instant) :=
  service j t >= job_cost j.
Definition job_response_time_bound (j : Job) (R : duration) :=
  completed_by j (job_arrival j + R).
```

We will in addition use a related definition from the `analysis` part, which introduces the notion of completion sequence.

▶ **Definition 6.** (`completion_sequence`) *Given an arrival sequence arrseq and a schedule sched, the corresponding completion sequence, denoted $endseq(arrseq, sched)$, is the function $\mathbb{N} \rightarrow 2^{Job}$ that maps each tick t to the jobs that complete at t.*

This is formalized in Coq using the following definition from `service.v`

```
Definition completes_at (j : Job) (t : instant) :=
    ~~ completed_by j t.-1 && completed_by j t.
```

and then in the file `completion_sequence.v` as

```
Definition completion_sequence : arrival_sequence Job :=
  fun t ⇒ [seq j ← arrivals_up_to arr_seq t | completes_at sched j t].
```

We have now all the basic concepts required to describe the behavior of a real-time system for RTA, except the notion of readiness, which is not needed in this paper. In the following, we will sometimes use the term *trace* to refer to a pair $(arrseq, sched)$.

[3] Note that we do not impose that a job receives exactly the amount of service corresponding to its cost. It could indeed receive more than needed to complete from the processor in the last tick of its execution.

4.2 Model

Unlike `behavior`, which is intended to be as universal as possible and to which all analyses using Prosa must relate, the `model` part is really meant as a library to be extended and picked from depending on the target system model and analysis. In this paper we will only use basic constructs from this part of Prosa to specify tasks and request bound functions.

Similar to jobs, a *task* in Prosa is nothing more than an abtract type with decidable equality. One usually specifies a task cost, and a function to relate jobs and tasks.

▶ **Definition 7.** (`job_task`, `task_cost`) *The task of a job j is denoted $task(j)$. The cost of a task tsk, of type* `work`*, is denoted $cost(tsk)$.*

This is formalized in Coq in the file `concept.v` of Prosa as

```
Definition TaskType := eqType.
Class JobTask (Job : JobType) (Task : TaskType) := job_task : Job -> Task.
Class TaskCost (Task : TaskType) := task_cost : Task -> duration.
```

Defining the arrival of a task is less trivial than for a job and there exists a variety of models in Prosa for doing so, including periodic and sporadic arrival models. Prosa also defines arrival curves, which however constrain the number of arrivals rather that the amount of requested workload as in NC. We use here request bound functions, which are closer to the NC arrival curves.

▶ **Definition 8.** (`request_bound`) *A request bound is a monotonic function $rbf : \mathbb{N} \to \mathbb{N}$ such that $rbf(0) = 0$. An arrival sequence $arrseq$ satisfies an upper request bound rbf for a given task tsk if for any ticks $t_1, t_2 \in \mathbb{N}$ such that $t_1 \leq t_2$, the cumulative cost of all jobs of tsk that arrive in $arrseq$ between t_1 and t_2 is bounded by $rbf(t_2 - t_1)$.*

This is formalized in Coq in the file `request_bound_functions.v` of Prosa as

```
Definition valid_request_bound_function (request_bound : duration -> work)
  := request_bound 0 = 0 ∧ monotone leq request_bound.

Definition respects_max_request_bound (tsk : Task) (max_request_bound :
    duration -> work) := ∀ (t1 t2 : instant), t1 <= t2 ->
  cost_of_task_arrivals arr_seq tsk t1 t2 <= max_request_bound (t2 - t1).
```

Let us now introduce the first definition that is not already part of Prosa. We specify the FIFO property, which guarantees that jobs complete in the order in which they arrive.

▶ **Definition 9.** (`FIFO_property`) *A trace $(arrseq, sched)$ is said to respect the FIFO property when for any two jobs j_1, j_2, if j_1 arrives before j_2 then it also completes before it, that is:*

$$\begin{aligned}&\forall j_1, j_2, \forall t_1, t_2, t_2',\\&\quad (j_1 \in arrseq(t_1) \land j_2 \in arrseq(t_2) \land t_1 < t_2 \land j_2 \in endseq(arrseq, sched)(t_2')) \implies\\&\qquad \exists t_1', t_1' \leq t_2' \land j_1 \in endseq(arrseq, sched)(t_1').\end{aligned}$$

For flexibility, we use a version of the FIFO property that applies to jobs of a specific pair of tasks. If it holds for all pairs of a task set then we end up with the above property. The advantage of this version is that it can be used to express the general FIFO property as well as the specific FIFO order between jobs of the same task (which is related to task sequentiality in Prosa).

▶ **Definition 10.** (`FIFO_per_task_property`) *A trace $(arrseq, sched)$ is said to respect the FIFO property with respect to two tasks tsk_1 and tsk_2 when any job j_2 in tsk_2 completes after all jobs j_1 arrived before it in tsk_1, that is:*

$$\begin{aligned}&\forall j_1, j_2, \forall t_1, t_2, t_2', (task(j_1) = tsk_1 \wedge task(j_2) = tsk_2 \wedge \\ &\quad j_1 \in arrseq(t_1) \wedge j_2 \in arrseq(t_2) \wedge t_1 < t_2 \wedge j_2 \in endseq(arrseq, sched)(t_2')) \implies \\ &\qquad \exists t_1', t_1' \leq t_2' \wedge j_1 \in endseq(arrseq, sched)(t_1').\end{aligned}$$

Note that there is a definition of FIFO in the model part of the Prosa library, which we do not use. First because it is quite complex, in the sense that it is derived from a more general notion of job level fixed priority. It is therefore more convenient to use a more straightforward definition for connecting RTA to NC. Formally relating our direct definition of FIFO to the Prosa definition is left for future work. Second, and more importantly, the Prosa FIFO definition is in fact different from ours, as it constrains the scheduling of jobs in a first-come-first-serve manner while the above definition constrains the order in which jobs complete. In particular, the Prosa FIFO scheduling does not necessarily guarantee the FIFO property in multicore systems. The notion of server in NC is very different from the notion of scheduler in RTA, and as a result a formal link between the two can only succeed at the interface, expressed in terms of arrivals and completions.

5 Network Calculus

Similarly to how the Prosa library is formalizing RTA analyses in Coq, the NC theory is formalized in the NCCoq library, which is available at `https://gitlab.mpi-sws.org/proux/nc-coq`. A short overview can be found in [22]. This section introduces the notions from NCCoq that we have used in our work.

5.1 Behavior

NC models dense time using the set of real numbers $\mathbb{R}$, and more specifically its subset of nonnegative values $\mathbb{R}_+ := \{x \in \mathbb{R} \mid x \geq 0\}$. Positive real numbers $\mathbb{R}_+^* := \{x \in R \mid x > 0\}$ as well as extended reals $\overline{\mathbb{R}} := \mathbb{R} \cup \{ -\infty, +\infty \}$ also play an important role. In the NCCoq formalization, these fundamental definitions are taken from the MathComp Analysis library [2] and respectively noted `R : realType`, `{nonneg R}`, `{posnum R}` and `\bar R`. NC relies on a few classes of functions, defined as follows.

▶ **Definition 11.** (`F`) $\mathcal{F} := \mathbb{R}_+ \to \overline{\mathbb{R}}$ *is the set of functions from* $\mathbb{R}_+$ *to* $\overline{\mathbb{R}}$*.*

This is formalized in Coq in the file `RminStruct.v` of NCCoq as

```
Definition F := {nonneg R} -> \bar R.
```

▶ **Definition 12.** (`Fplus`) $\mathcal{F}_+ := \{f \in \mathcal{F} \mid 0 \leq f\}$ *is the subset of functions from* $\mathcal{F}$ *that are nonnegative.*

This is formalized in Coq as

```
Definition F_0 : F := fun=> 0%E.
Definition nonNegativeF := [qualify a f | F_0 <= f ]%O.
Record Fplus := { Fplus_val :> F; _ : Fplus_val \is a nonNegativeF }.
```

where the first line defines $f_0 : \mathcal{F}$ the constant[4] function equal to $0 \in \overline{\mathbb{R}}$; the second line defines the nonnegative functions as functions that are larger than f_0. The third line uses a common Coq construction, where the subset of a set X satisfying a property P is defined as records with an element $x \in X$ of this set (here, `Fplus_val` of type[5] `F`) and an unnamed `_` proof of $P(x)$ (here, a proof that `Fplus_val` is a nonnegative function).

▶ **Definition 13.** (`Fup`) $\mathcal{F}^{\uparrow} := \{f \in \mathcal{F}_+ \mid \forall xy, x \leq y \Rightarrow f(x) \leq f(y)\}$ *is the subset of nondecreasing functions from* $\mathcal{F}_+$*.*

The Coq formalization `Fup` proceeds similarly to `Fplus`.

Now equipped with these basic definitions, we can define the main object of NC: data flow cumulative curves. They play a similar role to arrival sequences in RTA.

▶ **Definition 14.** (`flow_cc`) *A* cumulative curve *is a function* $f \in \mathcal{F}^{\uparrow}$ *satisfying*
- $f(0) = 0$
- f *is left continuous*
- f *only takes finite values:* $\forall t, f(t) \in \mathbb{R}_+$

We denote $\mathcal{C}$ *the set of cumulative curves.*

This is formalized in Coq in the file `cumulative_curves.v` of NCCoq.

NC defines delays using the notion of horizontal deviation between two cumulative curves.

▶ **Definition 15.** (`hDev_at`, `hDev`) *For* $f, g \in \mathcal{F}$ *and* $t \in \mathbb{R}_+$*, the horizontal deviation* $hDev(f, g, t) \in \overline{\mathbb{R}}_+$ *between* f *and* g *at* t *is defined as*

$$hDev(f, g, t) := \inf \{d \in \mathbb{R}_+ \mid f(t) \leq g(t + d)\}$$

and the horizontal deviation $hDev(f, g) \in \overline{\mathbb{R}}_+$ *between* f *and* g *is defined as*

$$hDev(f, g) := \sup \{hDev(f, g, t) \mid t \in \mathbb{R}_+\}.$$

This is formalized in Coq in the file `deviations.v` of NCCoq.

Finally, *servers* constitute the last main notion of NC to model concrete behaviors. NCCoq includes a few different flavors of servers. This notion would relate to the notion of scheduler in the RTA world. There is however is no such thing formalized in the core of Prosa, which is based on schedules in the `behavior` and on scheduling policies (that are predicates on traces) in the `model` part. We thus simply omit servers from the current paper, in which we will directly deal with input and output cumulative curves.

5.2 Model

Just like request bound functions are a tool to express contracts on arrivals in RTA, arrival curves are the NC tool to specify inputs of servers.

[4] $0 \in \overline{\mathbb{R}}$ is denoted `0%E` in Coq, `%E` being the scope annotation for extended real notations in the MathComp Analysis library [2].

[5] The `:>` syntax makes `Fplus_val` a Coq coercion, meaning `Fplus_val` will be inserted automatically by Coq to cast a `Fplus` as a `F` wherever needed. In practice, this means that a value of type `Fplus` can be used just like a function of type `F`.

▶ **Definition 16.** (`is_maximal_arrival`) *A function $\alpha \in \mathcal{F}$ is an* arrival curve *for any cumulative curve A when*

$$\forall t, d \in \mathbb{R}_+,\ A(t+d) - A(t) \leq \alpha(d)$$

This is formalized in Coq in the file `arrival_curves.v` of NCCoq.

This concludes our overview of the already existing definitions that are needed to present our contribution in the remainder of the paper.

6 Dense versus Discrete Time

Hardware clocks are devices whose aim is to provide a measure of time, generally based on a physical oscillator, characterized by its frequency f. Based on [16], we may distinguish ideal clocks, when the difference between two signal occurrences is exactly $1/f$, from constant drifted clocks, when the difference between two signal occurrences is exactly ρ/f, where $\rho - 1$ represents the drift (commonly related to the temperature), or more general constraints [25]. In computers, the time value is computed as a function based on a counter incremented at each signal occurrence.

Thus, Section 6.1 will introduce a universal notion of clock, to make a link between time and its discrete measurement, and Section 6.2 will introduce elements on the clock quality. The Coq definitions and lemmas referenced in this section can be found in the file `clock.v`.

6.1 Clocks

Since RTA relies on a discrete notion of time (ticks are in $\mathbb{N}$) whereas NC uses real times in $\mathbb{R}_+$, we need a link between discrete and dense times in order to relate both theories. A simple solution would be to use the canonical injection of $\mathbb{N}$ in $\mathbb{R}_+$, that is, to consider each tick $n \in \mathbb{N}$ as happening at real time $n \in \mathbb{R}_+$. However, doing this would preclude any modeling of behaviors such as clock drifts. We thus need a more generic modeling. To that end, we first introduce the notion of *universal clock*.

▶ **Definition 17.** (`uclock`) *A* universal clock *is a function $c : \mathbb{N} \to \mathbb{R}_+$ satisfying*

- $c(0) = 0$
- *there exists a $min\_intertick_c \in \mathbb{R}_+^*$ such that for all $n \in \mathbb{N}$, we have*

$$c(n+1) \geq c(n) + min\_intertick_c.$$

Thus given a clock c, the real value $c(n)$ is the physical time of the n-th clock tick. The condition $c(0) = 0$ means that the clock starts right away while the $min\_intertick_c$ is mostly there to exclude functions such as $n \mapsto 1 - \frac{1}{n}$ which could cause all kinds of Zeno phenomena.

While we now have a link from discrete to dense time, it would be useful to get some kind of inverse from dense to discrete time. One can notice that, thanks to the condition on $min\_intertick_c$ above, for any clock c and real time r, there is a tick n that happens exactly at time r or is the next one after r.

▶ **Lemma 18.** (`next_tick_ex`) *For any clock c and $r \in \mathbb{R}_+$, there exist $n \in \mathbb{N}$ such that $r \leq c(n)$ and for all $n' < n$, we have $c(n') < r$.*

This gives us a function $next_c : \mathbb{R}_+ \to \mathbb{N}$.

▶ **Definition 19.** (`next_tick`) *For any clock c, we denote $next_c : \mathbb{R}_+ \to \mathbb{N}$ the function mapping each r to the n given by Lemma 18.*

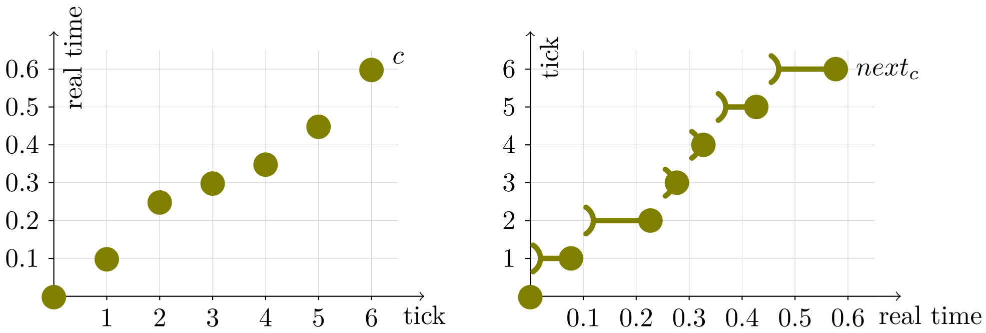

Figure 4 Graphical representation of an arbitrary clock c and its inverse function $next_c$.

This is illustrated in Figure 4 and formalized as follows in Coq: we first prove Lemma 18

```
Lemma next_tick_ex (c : uclock) (r : {posnum R}) :
  { n | r%:nngnum <= (c n)%:nngnum
    ∧ ∀ n', (n' < n)%N -> (c n')%:nngnum < r%:nngnum }.
```

where {n|P(n)} is a Coq notation for "there exists a n such that $P(n)$". Since such a proof is a simply dependent pair $(n, proof of P(n))$, one can use the first projection `proj1_sig` to extract n out of it.

```
Definition next_tick (c : uclock) (r : {posnum R}) : instant :=
  proj1_sig (next_tick_ex c r).
```

▶ Remark 20. We could as well have chosen n such that $r < c(n)$ rather than $r \leq c(n)$. The current choice appeared more convenient to match the left continuity conditions of NC.

Once clocks are defined, one can use Coq to formally verify a few lemmas about them. In practice, we have proved about a dozen lemmas that came useful in the remaining of our formal development, among which

▶ **Lemma 21.** *(`next_tick_0`) For any clock c, then $next_c(0) = 0$.*

▶ **Lemma 22.** *(`uclockK`) For any clock c, any tick $n \in \mathbb{N}$, then $next_c(c(n)) = n$.*

6.2 Pseudo Periodic Clocks

Although the condition on $min\_intertick_c$ in the definition of clocks was primarily introduced to rule out Zeno phenomena, it can also be used to model clocks whose time between two subsequent ticks is lower bounded. In addition, we will need to model clocks whose intertick time is also upper bounded. Such clocks can be seen as periodic clocks with an inexact period that can vary between some minimal and maximal value. They are thus called *pseudo periodic clocks*.

▶ **Definition 23.** (`ppuclock`) *A pseudo periodic clock is a clock c such that there exist $max\_intertick$ satisfying $min\_intertick_c \leq max\_intertick$ and for all n, we have $c(n+1) \leq c(n) + max\_intertick$.*

▶ **Example 24.** (`periodic_uclock`) Periodic clocks are a special case of pseudo periodic clocks for which $min\_intertick_c = max\_intertick$ and the n-th tick happens exactly at time $n \times period$.

```
Program Definition periodic_uclock (period : {posnum R}) : ppuclock :=
  Build_ppuclock
    (Build_uclock (fun n => (n%:R * period%:num)%:nng) _ period _)
    period _ _.
```

Thus, a clock c such that: $\forall n, c(n+1) - c(n) = 10^{-9}s$ is a periodic clock representing an ideal 1MHz clock, whereas a clock such that: $\forall n, c(n+1) - c(n) = (10^{-9} + 10^{-12})s$ is also a periodic clock, but representing a 1MHz clock with a constant drift of 0.1%. Finally, any clock such that: $\forall n, c(n+1) - c(n) \in [10^{-9} - 10^{-12}, 10^{-9} + 10^{-12}]$ is a pseudo periodic clock with a non constant drift not greater than 0.1%.

As a tool to link physical time and clock ticks, these definitions of clocks will be pervasive to link RTA and NC in the rest of this paper. Using different clocks for different parts of a system then enables to model drift between different physical clocks.

7 Linking Arrival Sequences and Cumulative Curves

Equipped with this notion of clock, we can now relate arrival sequences from RTA to cumulative curves from NC. We first do so for a single job then for an entire task.

The Coq definitions and lemmas referenced in this section can be found in the file `flow_cc_of_arrival_sequence.v`.

7.1 For a Single Job

▶ **Definition 25.** (`flow_cc_of_job`) *Given a job j and a clock c, one can build a cumulative curve $A_j \in \mathcal{C}$, as seen in Section 5, defined by*

$$A_j : \mathbb{R}_+ \to \mathbb{R}_+ \tag{1}$$

$$t \mapsto \begin{cases} 0 & \textit{if } t \leq c\,(arr(j)) \\ cost(j) & \textit{otherwise.} \end{cases} \tag{2}$$

We can thus establish a relation between jobs and cumulative curves.

▶ **Definition 26.** (`related_job_flow_cc`) *A job j, with its cost and arrival time, and a cumulative curve A are* related *when A is exactly the function A_j of Definition 25.*

7.2 For an Entire Task

We can proceed similarly for a task with multiple jobs.

▶ **Definition 27.** (`flow_cc_of_arrival_sequence`) *Given an arrival sequence arrseq, a clock c and a task tsk, the cumulative curve $A_{tsk} \in \mathcal{C}$ is defined by*

$$A_{tsk} : \mathbb{R}_+ \to \mathbb{R}_+ \tag{3}$$

$$t \mapsto \sum_{\substack{j \in \bigcup \{arrseq(i) \mid i < next_c(t)\}, \\ task(j) = tsk}} cost(j). \tag{4}$$

In Coq, we first define the arrival for a given task between two instants t_1 and t_2.

```
Let arrivals_of_tsk (arr_seq : arrival_sequence Job) tsk t1 t2 :=
  [seq j <- arrivals_between arr_seq t1 t2 | job_task j == tsk].
```

Then, given a clock c, we define the cumulative curve

```
Program Definition flow_cc_of_arrival_sequence
        (s : arrival_sequence Job) (c : uclock) (tsk : Task) :=
  Build_flow_cc (Build_Fup (Build_Fplus
    (fun t => (\sum_(j <- arrivals_of_tsk s tsk 0 (next_tick c t))
                 job_cost j)%:R%:E)
    _) _) _.
```

It is worth noting here that, although it may not be immediately apparent in the pen-and-paper Definition 27, this definition involves some proofs[6] to prove that A_{tsk} is actually a cumulative curve, i.e., is in $\mathcal{C}$.

We can thus establish a relation between arrival sequences and cumulative curves.

▶ **Definition 28.** (`related_arrival_flow_cc`) *Given a clock c and a task tsk, an arrival sequence arrseq and a cumulative curve A are* related *when A is exactly the function A_{tsk} of Definition 27.*

Finally, we can prove that the cumulative curve for a task is nothing else than the sum of the cumulative curves for each individual job in the task.

▶ **Lemma 29.** (`flow_cc_of_arrival_sequence_of_job`) *Given a clock c, a task tsk and an arrival sequence arrseq, we have for all $t \in \mathbb{R}_+$:*

$$A_{tsk}(t) = \sum_{\substack{j \in \bigcup \{arrseq(i) \mid i < next_c(t)\}, \\ task(j) = tsk}} A_j(t).$$

Note that we may want to write $A_{tsk}(t) = \sum_{j \in \bigcup \{ arrseq(i) \}, task(j) = tsk} A_j(t)$ since $A_j(t) = 0$ for jobs arriving after t, but it would lead to infinite sums whose convergence has to be proved to Coq. The given statement is equivalent and more convenient.

We now have a link between the main concrete behavior notions in RTA and NC, namely arrival sequences and cumulative curves. This link constitutes the basis enabling to relate other notions in the rest of the paper: notions of response time and delay, contracts with request bound functions and arrivals curves or various scheduling policies.

8 Linking Response Time and Horizontal Deviation

Equipped with a formal link between RTA's arrival sequences and NC's cumulative curves, we can now relate response times and horizontal deviations. Again, we first do so for a single job then for an entire task with multiple jobs.

The Coq definitions and lemmas referenced in this section can be found in the file `delay.v`.

[6] By filling the holes _ of `Program Definition`, the proofs themselves are present in the source file but omitted here for the sake of clarity.

8.1 For a Single Job

▶ **Definition 30.** (`arrival_sequence_of_job`) *Given a set of jobs Job and a job $j \in Job$, we define the arrival sequence $arrseq_j$ of this job as*

$$arrseq_j : \mathbb{N} \to 2^{Job} \tag{5}$$

$$t \mapsto \begin{cases} \{\, j \,\} & \text{if } t = arr(j) \\ \emptyset & \text{otherwise.} \end{cases} \tag{6}$$

We then prove that this arrival sequence is consistent with the considered `job_arrival` function, meaning that $j \in arrseq_j(arr(j))$.

▶ **Lemma 31.** (`arrival_sequence_of_job_consistent`) *The arrival sequence $arrseq_j$ of a job j, as defined in Definition 30, is consistent with the arrival time of j.*

Thus, from a response time bound on an arrival sequence, one can deduce a horizontal deviation on related arrival curves.

▶ **Lemma 32.** (`hDev_of_job_response_time`) *Given a pseudo periodic clock c, for a given job j, a related cumulative curve A and a cumulative curve D related to the completion sequence of j, if some $r^+ \in \mathbb{N}$ is a response time bound for j, then*

$$hDev(A, D) \leq r^+ \times max\_intertick_c.$$

Note that here, a pseudo periodic clock, and not just a clock, is required because the result involves the upper bound $max\_intertick$ between two consecutive clock ticks. A similar result holds in the reverse direction.

▶ **Lemma 33.** (`job_response_time_of_hDev`) *Given a clock c, for a given job j, a related cumulative curve A and a cumulative curve D related to the completion sequence of j, given a bound on the horizontal deviation between A and D, if the horizontal deviation between A and D is bounded by $r^+ \times min\_intertick_c$ for some $r^+ \in \mathbb{N}$*

$$hDev(A, D) \leq r^+ \times min\_intertick_c$$

then r^+ is a response time bound for j.

Note that this involves the bound $min\_intertick_c$ between two consecutive clock ticks.

8.2 For an Entire Task

We can proceed similarly for entire tasks. Thus, from a task response time bound on an arrival sequence, one can deduce an horizontal deviation on related arrival curves.

▶ **Lemma 34.** (`hDev_of_task_response_time`) *Given a pseudo periodic clock c, a task tsk, an arrival sequence $arrseq$, a related arrival curve A, a schedule $sched$ and a cumulative curve D related to its completion sequence $endseq(arrseq, sched)$, if some $r^+ \in \mathbb{N}$ is a response time bound for all jobs of task tsk, then*

$$hDev(A, D) \leq r^+ \times max\_intertick_c.$$

There is a big caveat for the reverse direction: since NC is unable to distinguish individual jobs, as cumulative curves only register the sum of their costs[7], one needs to add an additional FIFO hypothesis that the jobs are ordered the same way in the arrival and completion sequences. We end up with the following result to derive a RTA response time bound from a NC horizontal deviation.

▶ **Lemma 35.** (`task_response_time_of_hDev`) *Given a clock c, a task tsk, an arrival sequence $arrseq$, a related cumulative curve A, a schedule $sched$, and a cumulative curve D related to its completion sequence $endseq(arrseq, sched)$, if the FIFO property of Definition 10 is satisfied with $tsk_1 := tsk$ and $tsk_2 := tsk$ and if the horizontal deviation between A and D is bounded by $r^+ \times min\_intertick_c$ for some $r^+ \in \mathbb{N}$:*

$$hDev(A, D) \leq r^+ \times min\_intertick_c$$

then r^+ is a response time bound for all jobs of task tsk.

We now have a way to interpret RTA analyses results in a NC setting and vice versa.

9 Linking Request Bound Functions and Arrival Curves

The links established between RTA and NC in the previous sections were only pertaining to the *behavior* subsections of Sections 4 and 5, defining respectively RTA and NC. That is, those notions are purely mathematical, no actual computation is made on them. On the contrary, the notions of request bound functions (RBF) and arrival curves, defined in the *model* subsections of Sections 4 and 5, are actually manipulated by analyses. They are "computational" objects and not mere mathematical definitions. Thus, given a RBF, a "constructive" definition of a related arrival curve could enable to actually communicate results from a RTA analysis to a NC one, and vice versa. The current section precisely aims at building such "constructive" links.

The Coq definitions and lemmas referenced in this section can be found in the files `arrival_curve_of_request_bound_function.v` and `request_bound_function_of_arrival_curve.v`.

9.1 From Request Bound Functions to Arrival Curves

Given a RBF and a clock, one can define an arrival curve in $\mathcal{F}_+$.

▶ **Definition 36.** (`arrival_curve_of_request_bound_function`) *Given $rbf : \mathbb{N} \to \mathbb{N}$ and a clock c, we can define $\alpha_{rbf} : \mathcal{F}_+$ as*

$$\alpha_{rbf} : \mathbb{R}_+ \to \overline{\mathbb{R}}_+ \tag{7}$$

$$d \mapsto rbf\left(\left\lceil \frac{d}{min\_intertick_c} \right\rceil\right). \tag{8}$$

Moreover, if the RBF is valid, then the arrival curve can be proved to be in $\mathcal{F}^{\uparrow}$, i.e., it is nondecreasing.

▶ **Lemma 37.** (`arrival_curve_of_valid_request_bound_function`) *For any rbf and clock c, if rbf is a valid RBF (according to Definition 8), then $\alpha_{rbf} : \mathcal{F}^{\uparrow}$.*

[7] See the example in Section 3 with Figure 3.

One can then prove that our definition indeed translates RBFs into arrival curves.

▶ **Lemma 38.** (`arrival_curve_of_request_bound_function_is_maximal_arrival`) *Given a clock c, a task tsk, an arrival sequence $arrseq$, a cumulative curve A and a RBF rbf, then if $arrseq$ and A are related for task tsk, if rbf is a valid RBF for tsk in $arrseq$, then α_{rbf} is an arrival curve for A.*

It is worth noting that this correspondence is tight for periodic clocks (i.e., one can prove[8] the converse of Lemma 38) but can be conservative otherwise.

9.2 From Arrival Curves to Request Bound Functions

Conversely, from an arrival curve in $\mathcal{F}$, one can define a RBF.

▶ **Definition 39.** (`request_bound_function_of_arrival_curve`) *Given $\alpha : \mathcal{F}$ and a pseudo periodic clock c, we can define $rbf_\alpha : \mathbb{N} \to \mathbb{N}$ as*

$$rbf_\alpha : \mathbb{N} \to \mathbb{N} \tag{9}$$

$$d \mapsto \begin{cases} 0 & \text{if } d \leq 0 \\ \lceil \alpha\left(d \times max\_intertick_c\right) \rceil & \text{otherwise.} \end{cases} \tag{10}$$

Note that this requires a pseudo periodic clock as it involves $max\_intertick$. Also note that $rbf_\alpha(0)$ is explicitly set to 0 because RTA defines valid RBFs as starting at 0 whereas NC has no such requirement[9] on α. This definition is encoded as follows in Coq.

```
Definition request_bound_function_of_arrival_curve
           (alpha : F) (c : ppuclock) : duration -> nat :=
  fun d =>
    if (d <= 0)%N then 0%N else
      ‘|ceil (fine (alpha (d%:R * (ppuclock_max_intertick c)%:num)%:nng%R)
    )|%N.
```

There are two things worth noticing about that Coq statement that were not immediately apparent in Definition 39. First, there is an additional absolute value `‘|.|` around the ceiling function `ceil`. This is needed for the definition to typecheck because the ceiling function returns a (signed) integer whereas we need a natural number. In practice, since its argument is always nonnegative, it is a no-op. Second, we need to insert `fine` because $\alpha : \mathcal{F}$ returns values in $\overline{\mathbb{R}}$ whereas `ceil` expects an input in $\mathbb{R}$. `fine` acts as the identity function on $\mathbb{R}$ and maps infinites to 0. This will require an extra finiteness hypothesis in the next two lemmas.

When the arrival curve is in $\mathcal{F}^{\uparrow}$, the RBF can be proved valid.

▶ **Lemma 40.** (`valid_request_bound_function_of_arrival_curve`) *Given $\alpha : \mathcal{F}^{\uparrow}$ and a pseudo periodic clock c, if α only takes finite values (i.e., for all d, $\alpha(d) \in \mathbb{R}$), then rbf_α is a valid RBF.*

One can then prove that our definition indeed translates arrival curves to RBFs.

[8] C.f., lemma `arrival_curve_of_request_bound_function_respects_max_request_bound`.

[9] Even if, in practice, arrival curves with $\alpha(0) \neq 0$ are of no interest hence never used, the set $\mathcal{F}$ enjoys a nice algebraic structure of dioid which $\{\alpha : \mathcal{F} \mid \alpha(0) = 0\}$ doesn't and NC makes extensive use of this algebraic structure.

▶ **Lemma 41.** (request_bound_function_of_arrival_curve_respects_max_request_bound) *Given a pseudo periodic clock c, a task tsk, an arrival sequence $arrseq$, a cumulative curve A and $\alpha : \mathcal{F}^{\uparrow}$ such that for all d, $\alpha(d) \in \mathbb{R}$, then if $arrseq$ and A are related for task tsk, if α is an arrival curve for A, then rbf_{α} is a RBF for task tsk in s.*

Again this is tight only for periodic clocks.

The definitions of α_{rbf} and rbf_{α} do not appear directly computable as they act on an infinite domain but practical implementations of RTA or NC analyses usually handle some kind of periodic functions[10], in which case α_{rbf} and rbf_{α} can be actually computed. We believe this is a powerful result, offering an effective way to translate RTA results into NC hypotheses and vice versa. This enables us to combine the two theories and take advantage of their respective strengths to derive real-time analysis results that none of the techniques alone could provide.

10 Linking Scheduling Properties

We have already mentioned in Section 3 that the contribution does not address the scheduling itself. Nevertheless, the First In First Out (FIFO) property as given in Definition 10 is not a scheduling policy, but a property on schedulings. We are thus able to show equivalence of the FIFO property in both formalisms, as presented in Section 10.1. Moreover, while doing preliminary works toward an equivalence of static priority schedulings, an issue was uncovered, as presented in Section 10.2.

10.1 FIFO

While the definition of FIFO in the RTA setting, as provided in Definition 10, appears relatively straightforward, the NC definition may be much more enigmatic to anyone but NC experts. We prove that this NC definition matches the RTA one, thus giving it a higher confidence. The NC definition of the FIFO service policy is as follows.

▶ **Definition 42.** *For $n \in \mathbb{N}$, the cumulative curves A_i and D_i for $i < n$ are said to respect the FIFO service policy when*

$$\forall i, j \in \{0, \ldots, n-1\}, \forall t, u \in \mathbb{R}_+, A_i(u) < D_i(t) \implies A_j(u) \leq D_j(t).$$

The Coq definitions and lemmas referenced in this section can be found in the file `fifo.v`. We prove in particular the equivalence between the RTA and NC definitions of FIFO.

▶ **Lemma 43.** (FIFO_arrival_sequences_to_flow_cc) *Given a clock c, an arrival sequence $arrseq$, a schedule $sched$, two cumulative curves A and D respectively related to $arrseq$ and the completion sequence $endseq(arrseq, sched)$, if $arrseq$ and $sched$ respect the (RTA) FIFO property (for any pair of tasks, according to Definition 10), then A and D respect the (NC) FIFO policy (according to Definition 42).*

Note that for the converse to hold, we require the extra hypothesis that each task satisfies the FIFO policy with itself, because NC is blind on the order of jobs within a single task, as illustrated on Figure 3 in Section 3.

[10] For the very precise reason that it is possible to perform actual computations on this subclass of functions and that actual real-time behaviors are commonly periodic.

▶ **Lemma 44.** (`FIFO_flow_cc_to_arrival_sequences`) *Given a clock c, an arrival sequence arrseq, a schedule sched, two cumulative curves A and D respectively related to arrseq and the completion sequence endseq(arrseq, sched), if A and D respect the (NC) FIFO policy (according to Definition 42) and if for any task tsk, the arrival sequence arrseq and schedule sched satisfy the FIFO policy between task tsk and itself, then arrseq and sched respect the (RTA) FIFO policy (for any pair of tasks, according to Definition 10).*

Thus, we have seen that the definitions of FIFO in RTA and NC do match. This is a worthwhile result as its strengthens our confidence in both definitions. In particular, being more abstract, the NC definition easily looks rather mysterious for a non NC expert.

10.2 Fixed/Static Priority

Although the NC definition of static priority looks more natural than its definition of FIFO, we were eager to prove the same kind of equivalence for it. Here is the definition of fully preemptive static priority as given in [3, Def. 7.8]

▶ **Definition 45.** *Given $n \in \mathbb{N}$, the cumulative curves A_i and D_i for $i < n$ satisfy the static priority service policy when, for all $i < n$:*

$$\forall s, t \in \mathbb{R}_+, \left(\forall u \in [s, t], \sum_{j \prec i} A_j(u) > \sum_{j \prec i} D_j(u) \right) \implies D_i(t) = D_i(s)$$

where $\prec$ is a total order on $\{0, \ldots, n-1\}$.

In this definition, a flow j has a higher priority than i when $j \prec i$.

Equipped with this definition, the following lemma is proved.

▶ **Lemma 46.** (`FP_arrival_sequence_to_flow_cc`) *Given a clock c, an arrival sequence arrseq satisfying sequential readiness and a schedule sched on a fully preemptive ideal uniprocessor, a priority $\prec$, two cumulative curves A and D respectively related to arrseq and the completion sequence endseq(arrseq, sched), if arrseq and sched respect the (RTA) fixed priority policy relative to $\prec$, then A and D respect the (NC) static priority policy (according to Definition 45).*

The reverse doesn't hold since, as already explained, the departure curve D doesn't represent the scheduling *sched* but only the related completion sequence.

It is worth noting that the main theorem using Definition 45, given in [3, Thm. 7.6] is wrong[11] and was proved in NCCoq by strengthening the above definition, replacing the left-closed interval $[s, t]$ by the left-open one $(s, t]$. This small change was deemed innocuous at the time and did not raise further attention[12]. While attempting to prove the equivalence of the strengthened hypothesis with the RTA definition of the fully preemptive fixed priority policy, it became obvious that the hypothesis strengthening was not that innocuous as it broke the equivalence with RTA. Following this discovery, the theorem in NCCoq (`SP_residual_service_curve` in file `static_priority.v` of NCCoq) was rather fixed by strengthening another hypothesis, namely demanding the service curve of the aggregate server to be a cumulative curve[13] in $\mathcal{C}$.

[11] Counter example: consider two flows 1 and 2 with $1 \prec 2$, respective arrivals $A_1 := s_2$ and $A_2 := s_1$ and departures $D_1 := s_4$ and $D_2 := s_2$ (with s_d defined as $t \mapsto 0$ when $t \leq d$ and $t \mapsto 1$ otherwise) and an aggregated strict service β defined as $d \mapsto 0$ when $d < 2$ and $d \mapsto 1$ otherwise.

[12] Maybe because it was the most obvious hypothesis strengthening to make the proof given in [3] work.

[13] This hypothesis strengthening is reasonably innocuous as most service curves already live in $\mathcal{C}$ and could otherwise be easily under-approximated by a service curve in $\mathcal{C}$.

This experience report is particularly interesting as it shows how formal proofs of equivalence between two theories can unveil errors that where overlooked for a few years.

11 Conclusion

In this paper, we have built bridges between Response Time Analysis (RTA) and Network Calculus (NC) and formalized them with the Coq proof assistant. This shows how the notions of job, task, trace, response time, in RTA are related to the notions of arrival curve, departure curve and delay in NC. To do so, we have formalized a notion of (possibly drifty) clock and proved that what is called FIFO in both formalisms represents the same constraints. We also prove that bounds computed in one framework are valid in the other one, even when considering clock drifts.

The related work presented in Section 2 was already providing increased confidence into RTA and NC by achieving formal proofs in Coq of already known results, sometimes discovering bugs in proofs or in the results themselves. The new formal bridges between these two theories that coexist in the real-time community bring more than just confidence.

First, the most obvious contribution is a better mutual understanding between both communities. Such a comparison between different points of view may help each community in its reflection on its own models (e.g., NC has a notion of per-flow server, capturing several behaviors and not bound to a scheduling policy, whereas RTA has a notion of global trace, and sets of traces, but no scheduler).

Second, being able to go back and forth between theories allows us to analyze a complete system by using each theory where it is the most convenient and to combine the results to get the best of both theories in each component, like in [18] or more recently, in [17].

Finally, a third result, unexpected when we started this work, concerns the strength of modeling. Formal work makes it possible to check that some theory is correct, in the sense that the model fulfills some properties. But there is no way to check that the model reflects the reality. Building formal bridges between models is thus an effective way to increase our confidence in these models. For example, as reported in Section 10.2, this unveiled a weakness in the definition of static priority in the NC formal model.

These results open opportunities for future research. They provide a strong background for schedulability analyses in presence of clock drifts. They could also be pursued *per-se* to provide more links between RTA and NC, and also provide links with CPA.

Future work will consists in modeling the schedule itself. It may be possible for example to represent in NC the cumulative curves of release, schedule and completion of the job of Figure 1 as illustrated in Figure 5, inspired from [4, §2.3].

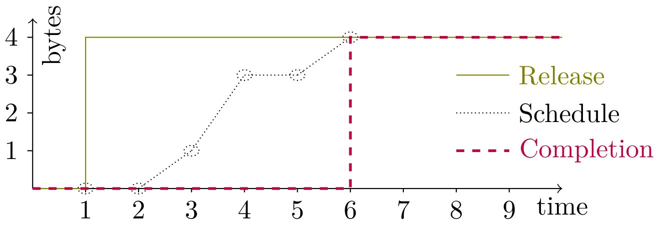

Figure 5 NC: Cumulative curves representing the job of Figure 1 and its schedule.

References

1 *28th Euromicro Conference on Real-Time Systems, ECRTS 2016, Toulouse, France, July 5-8, 2016*. IEEE Computer Society, 2016. URL: `https://ieeexplore.ieee.org/xpl/conhome/7557819/proceeding`.

2 Reynald Affeldt, Cyril Cohen, Marie Kerjean, Assia Mahboubi, Damien Rouhling, and Kazuhiko Sakaguchi. Competing inheritance paths in dependent type theory: a case study in functional analysis. In *IJCAR 2020 - International Joint Conference on Automated Reasoning*, pages 1–19, Paris, France, June 2020. URL: `https://hal.inria.fr/hal-02463336`.

3 Anne Bouillard, Marc Boyer, and Euriell Le Corronc. *Deterministic Network Calculus: From Theory to Practical Implementation*. John Wiley & Sons, Ltd, October 2018. `doi:10.1002/9781119440284`.

4 Anne Bouillard and Éric Thierry. An algorithmic toolbox for network calculus. *Discrete Event Dynamic Systems*, 18(1):3–49, October 2008. http://www.springerlink.com/content/876x51r6647r8g68/. `doi:10.1007/s10626-007-0028-x`.

5 Khaoula Boukir, Jean-Luc Béchennec, and Anne-Marie Déplanche. Requirement specification and model-checking of a real-time scheduler implementation. In Liliana Cucu-Grosjean, Roberto Medina, Sebastian Altmeyer, and Jean-Luc Scharbarg, editors, *28th International Conference on Real Time Networks and Systems, RTNS 2020, Paris, France, June 10, 2020*, pages 89–99. ACM, 2020. `doi:10.1145/3394810.3394817`.

6 Marc Boyer and Pierre Roux. Embedding network calculus and event stream theory in a common model. In *Proc. of the 21st IEEE Int. Conf. on Emerging Technologies and Factory Automation (ETFA 2016)*, Berlin, Germany, September 2016. `doi:10.1109/ETFA.2016.7733565`.

7 Sergey Bozhko and Björn B. Brandenburg. Abstract response-time analysis: A formal foundation for the busy-window principle. In Marcus Völp, editor, *32nd Euromicro Conference on Real-Time Systems, ECRTS 2020, July 7-10, 2020, Virtual Conference*, volume 165 of *LIPIcs*, pages 22:1–22:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. `doi:10.4230/LIPIcs.ECRTS.2020.22`.

8 Felipe Cerqueira, Geoffrey Nelissen, and Björn B. Brandenburg. On Strong and Weak Sustainability, with an Application to Self-Suspending Real-Time Tasks. In Sebastian Altmeyer, editor, *30th Euromicro Conference on Real-Time Systems (ECRTS 2018)*, volume 106 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 26:1–26:21, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. `doi:10.4230/LIPIcs.ECRTS.2018.26`.

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

10 The Coq development team. *The Coq proof assistant reference manual*, 2020. Version 8.13. URL: `https://coq.inria.fr`.

11 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, Montreal, QC, Canada, April 16-18, 2019*, pages 182–191, 2019. `doi:10.1109/RTAS.2019.00023`.

12 Pascal Fradet, Maxime Lesourd, Jean-François Monin, and Sophie Quinton. A generic coq proof of typical worst-case analysis. In *2018 IEEE Real-Time Systems Symposium, RTSS 2018, Nashville, TN, USA, December 11-14, 2018*, pages 218–229. IEEE Computer Society, 2018. `doi:10.1109/RTSS.2018.00039`.

13 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 Kimberly Keeton and Timothy Roscoe, editors, *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016*, pages 653–669. USENIX Association, 2016. URL: `https://www.usenix.org/conference/osdi16/technical-sessions/presentation/gu`.

14 Xiaojie Guo, Maxime Lesourd, Mengqi Liu, Lionel Rieg, and Zhong Shao. Integrating Formal Schedulability Analysis into a Verified OS Kernel. In *Computer Aided Verification*, pages 496–514, New York, United States, July 2019. doi:10.1007/978-3-030-25543-5_28.

15 Xiaojie Guo, Lionel Rieg, and Paolo Torrini. A generic approach for the certified schedulability analysis of software systems. In *RTCSA 2021 - 27th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 1–10, Houston (online), United States, August 2021. URL: https://hal.archives-ouvertes.fr/hal-03540548.

16 ITU-T. Definitions and terminology for synchronization networks. Technical Report Recommendation G.810, International telecommunication union (ITU), 1996.

17 Leonie Köhler, Borislav Nikolić, and Marc Boyer. Increasing accuracy of timing models: From cpa to cpa+. In *Proc. of the Design, Automation and Test in Europe Conference and Exhibition (DATE)*, Florence, Italy, March 2019.

18 Simon Künzli, Arne Hamann, Rolf Ernst, and Lothar Thiele. Combined approach to system level performance analysis of embedded systems. In Soonhoi Ha, Kiyoung Choi, Nikil D. Dutt, and Jürgen Teich, editors, *Proceedings of the 5th International Conference on Hardware/Software Codesign and System Synthesis, CODES+ISSS 2007, Salzburg, Austria, September 30 - October 3, 2007*, pages 63–68. ACM, 2007. doi:10.1145/1289816.1289835.

19 Xavier Leroy. Formal verification of a realistic compiler. *Commun. ACM*, 52(7):107–115, 2009. doi:10.1145/1538788.1538814.

20 Etienne Mabille, Marc Boyer, Loïc Fejoz, and Stephan Merz. Towards certifying network calculus. In *Proc. of the 4th Conference on Interactive Theorem Proving (ITP 2013)*, Rennes, France, July 2013.

21 Simon Perathoner, Ernesto Wandeler, Lothar Thiele, Arne Hamann, Simon Schliecker, Rafik Henia, Razvan Racu, Rolf Ernst, and Michael González Harbour. Influence of different system abstractions on the performance analysis of distributed real-time systems. In *Proceedings of the 7th ACM & IEEE international conference on Embedded software (EMSOFT'07)*, pages 193–202, New York, NY, USA, 2007. ACM. doi:10.1145/1289927.1289959.

22 Lucien Rakotomalala, Marc Boyer, and Pierre Roux. Formal Verification of Real-time Networks. In *JRWRTC 2019, Junior Workshop RTNS 2019*, TOULOUSE, France, November 2019. URL: https://hal.archives-ouvertes.fr/hal-02449140.

23 Lucien Rakotomalala, Pierre Roux, and Marc Boyer. Verifying min-plus computations with coq. In Aaron Dutle, Mariano M. Moscato, Laura Titolo, César A. Muñoz, and Ivan Perez, editors, *NASA Formal Methods - 13th International Symposium, NFM 2021, Virtual Event, May 24-28, 2021, Proceedings*, volume 12673 of *Lecture Notes in Computer Science*, pages 287–303. Springer, 2021. doi:10.1007/978-3-030-76384-8_18.

24 Jonas Rox and Rolf Ernst. Compositional performance analysis with improved analysis techniques for obtaining viable end-to-end latencies in distributed embedded systems. *International Journal on Software Tools for Technology Transfer*, 15(3):171–187, 2013.

25 Ludovic Thomas and Jean-Yves Le Boudec. On time synchronization issues in time-sentive networks with regulators and nonideal clocks. *Proceedings of the ACM on Measurement and Analysis of Computing SystemsJune 2020 Article No.: 27*, 4(2), June 2020. doi:10.1145/3392145.

26 Makarius Wenzel. *The Isabelle/Isar Reference Manual*, 2021. Version 2021-1. URL: https://isabelle.in.tum.de.