Formal Verification of Real-time Networks

Lucien Rakotomalala, Marc Boyer, Pierre Roux

Formal Verification of Real-time Networks

Lucien Rakotomalala Marc Boyer Pierre Roux
ONERA / DTIS
Université de Toulouse
F-31055 Toulouse – France
firstname.name@onera.fr

ABSTRACT

Embedded real-time networks must ensure guaranteed delays. Network calculus is a theory providing bounds on such delays. This mathematical theory currently relies on, human made, pen and paper proofs. The current work offers to formalize such proofs in Coq, an automated proof checker. We formalize a subset of the theory large enough to handle a complete proof of bounds on a representative case study.

Keywords

network-calculus, Coq, real-time network

1. INTRODUCTION

Nowadays, real-time systems are pervasive in embedded applications such as the aerospace or automotive industries. Such applications being critical, it is mandatory to establish a high degree of confidence in their functional and temporal behaviour. Whereas a lot of work is available on functional verification, this paper focuses on temporal correctness. Analysis methods in this regard do exist and they are mathematically proved. However, these proofs are only reviewed and verified by humans which implies a substantial risk of error, due to their complexity or subtle hypotheses.

Therefore, some mistakes can be made during the writing and reviewing process of a proof. A major source of mistakes is the omission of an implicit hypothesis when reusing a previous results. Such omissions have occurred several times in real-time analysis proofs. For example, it has been recently discovered that some self-suspension consideration was inexact 20 years after publication of the original paper [14]. As another example, an error in real-time analyses of the CAN bus, was discovered only 13 years after the original publication [9].

We aim at increasing the confidence in mathematical proofs by automating the proofreading process. This can be made by a computer running a proof assistant. Such tools are developed by computer scientists and mathematicians for nearly half a century. We can for instance mention Coq, Isabelle or PVS [3, 15, 16]. We can use one of them to formally define mathematical objects, enunciate theorems and finally describe proofs of these theorems. A computer is then able to automatically check these proofs.

As a first advantage to the use of a proof assistant, the confidence in the correctness of the proofs is reduced to the absence of bugs in the tool, the coherence of the implemented logic and potential axioms used. On this last point, the tool allows to know exactly which axioms are used in each proof.

Another advantage is to identify where and how hypotheses are used by a proof. In extreme cases, it happens that some hypotheses are in fact unused in the course of a proof. To check this, it is enough to remove the considered hypothesis and attempt a recompilation of the proof.

Finally, a proof assistant enables a simpler and safer reuse of the results: an application of a theorem is only possible when all hypotheses are collected. Thus, it is not possible to forget hypotheses.

The proof assistant we use in this paper is Coq [3], developed by Inria, based on a small kernel and extended by a large sets of libraries.

The kernel implements an intuitionistic logic. Our confidence is based on this reduced kernel that ultimately check all Coq proofs. This kernel uses a low level language that is simple but hardly usable by humans. Coq thus provides an interface to make it operable, it interprets user's commands to elaborate a proof in the kernel language.

Coq comes with a standard library providing mathematical models and properties. Other libraries go beyond this standard set, such as the Mathcomp [12] or the Coquelicot [4] libraries. The correctness of these libraries relies on the fact that they are checked by the kernel.

In order to formalize proofs on a specific problem, a Coq user first defines a mathematics model (the modeling of the problem). Then she expresses some properties of the model (stating lemmas or theorems) and eventually writes Coq commands to prove these properties.

During this process, Coq checks that definitions and statements of properties are well formed and that the proofs hold.

Our proofs will focus on embedded networks and will use an analysis method of temporal properties on these networks: the network calculus (NC). This theory heavily relies on tropical algebra through the dioid of *min-plus* functions.

As previously described, our first step will consist in writing NC definitions in the Coq language. Secondly, NC results found in the literature will be prove within Coq. Only a NC expert can check that the Coq definitions match with the ones in the literature. Thus our models have to be readable even without a deep Coq expertise. In contrast, the second step, proof writing, does not need to be checked by a human since the compilation guarantees the proofs.

While our final goal is to be able to verify a full industrial network, our current contributions are:

- an extension of the Coq Mathematical Components library of algebraic structures,
- formal definitions and proofs of some typical network cal-

culus theorems,

- an application on a first case study, handling 5 flows through 5 servers with FIFO policy.

Section 2 presents related works on proof assistants and real-time network analyses. Then, Section 3 presents our formal development. Section 4 is a case study on a performances analyses of a network. Finally, we conclude with Section 5 and explain our future work in Section 6.

2. RELATED WORK

Network calculus tools take as input the description of a network and compute delay bounds. The validity of these bounds relies on both the correctness of the network calculus theorems (produced by authors and checked by reviewers) and the correctness of the implementation (relying on developer skills).

Formally proving correct implementation was the aim of [11], using the proof assistant *Isabelle*. Our goal is to prove both theory and implementation correctness, using another proof assistant, Coq.

Our work is part of the project *RT-proofs* [2]. The main objective of this project is to lay the foundations for computer-assisted formal verification of timing analysis results. Many works have been performed already, for example a verification of a CAN schedulability analysis with Coq [10].

Finally, a library for the development of machine-checked schedulability analysis using Coq is also available [8]

3. NETWORK CALCULUS WITH COQ

The Network Calculus theory is based on the *min-plus* dioid [5]. Thus, our first contribution consists in adding this algebraic structure to the Mathcomp library (Sections 3.1, 3.2). We then formalize main NC results (Section 3.3). Some metrics on the Coq development are given in Section 3.4.

3.1 Algebraic structures

We use some of the existing elements in the Mathcomp library [12] to define the algebraic structure of complete dioids. The Mathcomp library provides some algebraic structures useful in our case (monoids, rings,...) but not dioids.

So, with the help of this library, we add a description of the dioid structure as defined by [13].

DEFINITION 1 (DIOID). *A set $\mathcal{D}$ with two operators $\oplus$ and $\otimes$ is called a dioid if*

- *$\oplus$ is associative and commutative and admits a neutral element $\bar{0}$*
- *$\otimes$ is associative and admits a neutral element $\bar{1}$*
- *$\otimes$ is left and right distributive over $\oplus$*
- *$\bar{0}$ is absorbing for $\otimes$*
- *$\oplus$ is idempotent, i.e: $\forall a \in \mathcal{D}, a \oplus a = a$*

A dioid is said to be complete if it is closed for infinite sum and if the product distributes over infinite sums on both sides.[1] Under some assumptions, a subset of a complete dioid remains a complete dioid.

[1]All these algebraic structures and their properties have been developed using only intuitionistic logic. Even proofs on infinite sums don't require classical reasoning since we only prove results based on the hypothesis that such sums do exist. In contrary, the next section will deal with real numbers whose formalization in the Coq standard library requires classical logic with the excluded middle axiom.

THEOREM 1. *Let $\mathcal{D}$ be a complete dioid with operators $\oplus$ and $\otimes$. Any $\mathcal{D}' \subseteq \mathcal{D}$ including $\bar{0}$ and $\bar{1}$ and stable for $\oplus$, $\otimes$ and infinite sums is also a complete dioid.*

We can then define the kleene operator, useful in NC.

DEFINITION 2 (KLEENE OPERATOR). *Let $\mathcal{D}$ be a complete dioid with operators $\oplus$ and $\otimes$, the kleene operator on $a \in \mathcal{D}$ is: $a^* = \bigoplus_{i=0}^{+\infty} a^i$ with $a^0 = \bar{1}$ and $a^{i+1} = a \otimes a^i$*

By using this definition, we can state the next theorem. This result is shared with language theory.

THEOREM 2. *Let $\mathcal{D}$ be a complete dioid with $\oplus$ and $\otimes$, for all a, b in $\mathcal{D}$, $a^* \otimes b$ is the least solution of $x = (a \otimes x) \oplus b$.*

There exist many other results on dioids [5, 13]. Among them we prove 78 properties useful for NC.

All of these results have been submitted as a pull request on the Mathcomp library[2].

3.2 Instances

To use these properties in the NC context, we have to prove that sets of interest are dioids. This implies to:

- give a set $\mathcal{D}$,
- give operators $\oplus$, $\otimes$ and their neutral elements,
- prove that dioid properties (cf. Definition 1: associativity, commutativity...) hold.

NC handles functions on real values: $\mathcal{F} : \mathbb{R}^+ \to \overline{\mathbb{R}}$, with $\overline{\mathbb{R}} = \mathbb{R} \bigcup \{-\infty, +\infty\}$, and uses the following two operators:

- minimum: $(f \wedge g)(t) = \min(f(t), g(t))$
- convolution: $(f * g)(t) = \inf_{0 \leqslant s \leqslant t} \{f(t-s) + g(s)\}$

THEOREM 3. *The set of functions $\mathcal{F}$ with $\oplus = \wedge$ and $\otimes = *$ is a complete dioid.*

Depending on the authors and even on the papers, NC results handle either $\mathcal{F}$ or some specific subsets:

- $\mathcal{F}^+ : \mathbb{R}^+ \to \overline{\mathbb{R}}^+$,
- $\mathcal{F}^\uparrow$: subset of non-decreasing elements of $\mathcal{F}^+$.

THEOREM 4. *The sets $\mathcal{F}^+$ and $\mathcal{F}^\uparrow$ with $\oplus = \wedge$ and $\otimes = *$ are complete dioids.*

This is proved by using Theorem 1. One contribution of our work is to explicit which subset is needed for each result.

To develop these constructions, we use results of the Coquelicot library [4]: the set $\overline{\mathbb{R}}$ and its properties.

3.3 Network calculus

3.3.1 Model

NC models data flows by the cumulative amount of data at a point in a network at time t.

DEFINITION 3 (CUMULATIVE FUNCTION). *A function f : $\mathbb{R}^+ \to \mathbb{R}^+$ is a cumulative function if f:*

- *is non-decreasing: $\forall t, d \in \mathbb{R}^+, f(t) \leqslant f(t+d)$,*
- *starts at 0: $f(0) = 0$,*
- *is left-continuous.*

[2]`https://github.com/math-comp/math-comp/pull/357`

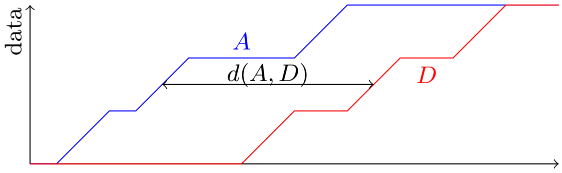

Figure 1: Illustration of the notion of delay

The set of cumulative functions is denoted $\mathcal{C}$. They are non-decreasing because they represent a cumulative amount of data and this one can not decrease. We consider that flows do not contain data before $t = 0$, so $\mathcal{C}$ functions start in 0.

A server is a relation between two cumulative functions: A for arrival and D for departure of the server.

Definition 4 (Server). *A server $S \subseteq \mathcal{C} \times \mathcal{C}$ satisfies:*

- $\forall A \in \mathcal{C},\ \exists D \in \mathcal{C},\ (A, D) \in S$
- $(A, D) \in S \Rightarrow D \leqslant A$

The first property means that, for all arrival there exists a departure. The second property means that departure can not happen before arrival so $D \leqslant A$.

Another notion of NC we use is called arrival curve. It is used to constrain cumulative functions.

Definition 5 (Arrival curve). *Let $A \in \mathcal{C}$ be a cumulative function. The function $\alpha \in \mathcal{F}^{\uparrow}$ is an arrival curve for A if $A \leqslant A * \alpha$.*

To specify servers, NC uses the notion of minimal service. We define it below, first using mathematics, then in Coq.

Definition 6 (minimal service). *Let S be a server and $\beta \in \mathcal{F}^{+}$. The server S is said to offer a* minimal service *curve β if: $\forall (A, D) \in S \implies A * \beta \leqslant D$.*

```
Definition is_min_service (S : C → C → Prop) (beta : Fplus)
          := forall A D, S A D → A ∗ beta ⩽ D.
```

The notation `Fplus` represents the set $\mathcal{F}^{+}$ presented in Section 3.2. (`S : C → C → Prop`) means that S is a relation on $\mathcal{C}$. The term `S A D` signifies $(A, D) \in S$ and the arrow $\rightarrow$ stands for logical implication in Coq.

3.3.2 Properties

To analyze temporal network performances, NC defines a notions of delay. The delay experienced by the flow whose arrival is A and departure is D is denoted $d(A, D)$, illustrated in figure 1. A formal definition can be found in [5].

Different policies for servers are defined in NC. One of them is the First-In First-Out policy. In such a server, each packet is served after all previously arrived packets have been served. For this policy, a NC theorem bounds the delay experienced by each packet. We proved this theorem in Coq.

Finally, NC provides a method to compute a contract on the output of a server,i.e., an arrival for the next server.

3.4 Coq development overview

Table 1 gives some metrics on our Coq development. It can first be observed that the number of definitions in *Instances* is higher compared to *Dioid* and *NC*. This difference has no significant meaning: it comes from a difference in Coq programming style between the two libraries.

	Definitions	Properties	Lines	Lines/prop.
Dioid	19	78	1616	21
Instances	108	159	2888	18
NC	26	52	2253	43

Table 1: Summary Table of Coq definitions and property done

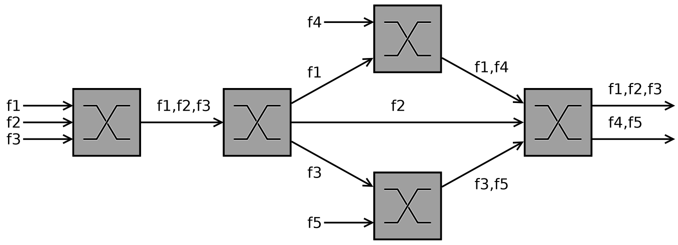

Figure 2: Network Topology

More interestingly, the ratio between the number of lines and the number of properties in *Dioid* and *Instances* is lower than in *NC*. This means that *Dioid* and *Instances* contain properties which only require short proofs, never exceeding 10 lines and most of time taking only one. On the contrary, *NC* requires larger, more complex, proofs.

These definitions and properties are very much inspired from a NC textbook [5], giving pen and paper proofs of NC properties. It is thus possible to compare pen and paper to Coq versions of these proofs. Formalizing pend and paper proofs in Coq provides several benefits. As expected, some typos have been found but we also found a few mistakes in proofs, which we had to fix[3]. More interestingly, some results appeared to be over specified: Coq helped us to simplify the hypotheses. Lastly, some properties have been both simplified and generalized: pen and paper proofs valid only for specific dioid instances have been leveraged to any dioid, with a shorter proof.

4. CASE STUDY

In this section, we work on a simple network with a particular topology shown on Figure 2. In this network, we consider that data transmissions are periodic with a 20 Mbits per second rate. The frame sizes are fixed to 1 Kbyte The scheduling policy for each server is FIFO, as introduced in subsection 3.3.2. The speed rate of each server is fixed to 100 Mbits per second. Servers are assumed to have no latency. Flows 1, 2 and 3 converge on the leftmost switch and share its output port. The next switch separate them: flow 1 goes up and competes with flow 4 on the output port of the upper switch. Flow 3 symmetrically goes down and meets the flow 5. All flows then converge to the rightmost switch, flows 1, 2 and 3 sharing one output port and flows 4 and 5 the other.

The objective is, for each flow, to bound the delay when crossing the entire network. To do so, we have written a Coq proof which consists in two steps. A first step considers each server and its crossing flows individually, and applies the Coq results presented in the previous sections. The second step consists in combining local results with respect to

[3]For example, in proof of theorem 6.2, a confusion was made between $\geqslant$ and $>$, invalidating the proof.

Flow	1	2	3	4	5
Delays bound (μs)	601.6	368	601.6	233.6	233.6

Table 2: Delay bound for each flow

the topology presented in Figure 2. This leads to algebraic expressions of the delays (in the *min-plus* dioid) whose numerical values are computed using the *min-plus* calculator from RTaW [1]. This tool implements the algorithms from [6] whose pen and paper proofs have not been formalized in Coq yet. The results are presented in Table 2.

5. CONCLUSION

The aim of this work was to formalize (using Coq) results on delay bounds of real-time network (using the NC theory).

This required the formalization in Coq of the algebraic structure of complete dioids. We rely on the Mathcomp library and we shared our development to this library. Then, we built specific instances of complete dioids used in NC with the help from the Coquelicot library.

Last, we developed a set of NC definitions and results, sufficient to perform the complete proof of a first case study.

Thus, we obtained a Coq development of 6757 lines containing a definition of the algebraic structure of dioids, instances of dioids and NC results. This work took one year, considering that the main author was a newcomer to both Coq and the NC theory.

Several benefits come with this formal development: we found a few mistakes in proofs from [5], which we had to fix. More interestingly, some results appeared to be over specified: Coq helped us to reduce the hypotheses. Last, some results have been generalized while simplifying their proofs.

Finally, the results are applied to a first case study. We used here a tool from RTaW to compute the final numerical results but Coq is used to prove the correctness of the computed expressions and all properties used to obtain these expressions.

We notice that there are three possible kinds of modifications of our case study. First, a modification of its numerical values (throughput, packet sizes...) does not change the Coq proof, since only numerical parameters of the final computation are affected. Second, a modification of the service policy requires to prove new theorems related to the new policy, but does not change the global structure of the proof. Finally, a modification in the network topology or routing breaks the structure of the proof.

6. FUTURE WORK

One may wonder how the work done for this small case study is relevant for realistic configurations.

Verification of actual embedded network, like AFDX [7] requires only two more results: on static priority scheduling and packetisation. We plan to add such Coq proofs.

In our case study, we use an external tool to compute the value of analytic expressions. We plan to either have Coq compute them by itself or verify the values computed by the external tool. This will allow us to have a complete Coq validation of performances bounds values.

The change of routing implies a manual modification of the proof. However, the structure of the proof is very repetitive, and quite a direct mapping of the network topology. We plan to automatize this part: either inside Coq, using dedicated tactics, or collaborating with an external tool, as done in [10].

7. REFERENCES

[1] Real time at work. `http://realtimeatwork.com/`.

[2] RT-proofs main page. `https://rt-proofs.inria.fr/`.

[3] Y. Bertot and P. Castéran. *Interactive theorem proving and program development. Coq'Art: The Calculus of inductive constructions.* 01 2004.

[4] S. Boldo, C. Lelay, and G. Melquiond. Coquelicot: A user-friendly library of real analysis for coq. *Math. in Computer Science*, 9(1):41–62, Mar 2015.

[5] A. Bouillard, M. Boyer, and E. Le Corronc. *Deterministic Network Calculus: From Theory to Practical Implementation.* 10 2018.

[6] A. Bouillard and E. Thierry. An algorithmic toolbox for network calculus. *Discrete Event Dynamic Systems: Theory and Applications*, 18, 03 2008.

[7] M. Boyer, N. Navet, and M. Fumey. Experimental assessment of timing verification techniques for AFDX. In *6th European Congress on Embedded Real Time Software and Systems*, Toulouse, France, Feb. 2012.

[8] F. Cerqueira, F. Stutz, and B. B. Brandenburg. PROSA: A case for readable mechanized schedulability analysis. In *28th Euromicro Conference on Real-Time Systems, ECRTS 2016, Toulouse, France, July 5-8, 2016*, pages 273–284, 2016.

[9] R. I. Davis, A. Burns, R. J. Bril, and J. J. Lukkien. Controller area network (can) schedulability analysis: Refuted, revisited and revised. *Real-Time Systems*, 35(3):239–272, Apr 2007.

[10] P. Fradet, X. Guo, J.-F. Monin, and S. Quinton. CertiCAN: A Tool for the Coq Certification of CAN Analysis Results. In *RTAS 2019 - 25th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 1–10, Montreal, Canada, Apr. 2019.

[11] E. Mabille, M. Boyer, L. Fejoz, and S. Merz. Towards certifying network calculus. In *Interactive Theorem Proving, Rennes, France, July 22-26*, 2013.

[12] A. Mahboubi and E. Tassi. *Mathematical Components.* 2018.

[13] M. Minoux and M. Gondran. *Graphs, Dioids and Semirings. New Models and Algorithms*, volume 41 of *Operations Research/Computer Science Interfaces Series*. Springer, 2008.

[14] G. Nelissen, J. Fonseca, G. Raravi, and V. Nélis. Timing analysis of fixed priority self-suspending sporadic tasks. In *2015 27th Euromicro Conference on Real-Time Systems*, pages 80–89, July 2015.

[15] S. Owre, J. M. Rushby, , and N. Shankar. PVS: A prototype verification system. In D. Kapur, editor, *11th International Conference on Automated Deduction (CADE)*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 748–752, Saratoga, NY, jun 1992. Springer-Verlag.

[16] M. Wenzel, L. C. Paulson, and T. Nipkow. The isabelle framework. In O. A. Mohamed, C. Muñoz, and S. Tahar, editors, *Theorem Proving in Higher Order Logics*, pages 33–38, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.