Extending the Network Calculus Algorithmic Toolbox for Ultimately Pseudo-Periodic Functions: Pseudo-Inverse and Composition

Raffaele Zippo[1,2,3*], Paul Nikolaus[3] and Giovanni Stea[2]

[1*]Dipartimento di Ingegneria dell'Informazione, Università di Firenze, Via di S. Marta 3, Firenze, 50139, Italy.
[2]Dipartimento di Ingegneria dell'Informazione, Università di Pisa, Largo Lucio Lazzarino 1, Pisa, 56122, Italy.
[3]Distributed Computer Systems Lab (DISCO), TU Kaiserslautern, Paul-Ehrlich-Straße 34, Kaiserslautern, 67663, Germany.

*Corresponding author(s). E-mail(s): raffaele.zippo@unifi.it; Contributing authors: nikolaus@cs.uni-kl.de; giovanni.stea@unipi.it;

Abstract

Network Calculus (NC) is an algebraic theory that represents traffic and service guarantees as curves in a Cartesian plane, in order to compute performance guarantees for flows traversing a network. NC uses transformation operations, e.g., min-plus convolution of two curves, to model how the traffic profile changes with the traversal of network nodes. Such operations, while mathematically well-defined, can quickly become unmanageable to compute using simple pen and paper for any non-trivial case, hence the need for algorithmic descriptions. Previous work identified the class of piecewise affine functions which are ultimately pseudo-periodic (UPP) as being closed under the main NC operations and able to be described finitely. Algorithms that embody NC operations taking as operands UPP curves have been defined and proved correct, thus enabling software implementations of these operations. However, recent advancements in NC make use of operations, namely the *lower pseudo-inverse*, *upper pseudo-inverse*, and *composition*, that are well-defined from an algebraic standpoint, but whose algorithmic aspects have not been addressed yet. In this paper, we introduce algorithms for the above operations when operands are UPP curves, thus extending the available algorithmic toolbox for NC. We discuss the algorithmic properties of these operations, providing formal proofs of correctness.

Keywords: Network calculus, min-plus algebra, algorithms, pseudo-inverse, composition

1 Introduction

Network Calculus (NC) is an algebraic theory where traffic and service guarantees are represented as functions of time. The I/O transformations that network traversal imposes on an input traffic can be represented as operations of min-plus algebra involving these curves. This allows one to compute worst-case performance guarantees for a flow traversing a network. NC dates back to the early 1990s, and it is mainly due to the work of Cruz [1, 2], Le Boudec and Thiran [3], and Chang [4]. Originally devised for the Internet, where it was used to engineer models of service [5–9], it has found applications in several other contexts, from sensor networks [10] to avionic networks [11, 12], industrial networks [13–15], automotive systems [16], and systems architecture [17, 18].

NC characterizes constraints on traffic arrivals (due to traffic shaping) and on minimum received service (due to scheduling) as *curves*, i.e., functions of time.[1] These curves are then used with operators from min-plus and max-plus algebra to derive further insights about the system. For example, the per-flow service curve of a scheduler, such as Weighted Round Robin, or performance bounds on the traffic such as an end-to-end delay bound. While these operations can be computed with pen and paper for simple examples, in most practical cases the application of NC requires the use of software. To this end, works [19, 20] provide an "algorithmic toolbox" for NC: they show that piecewise affine functions that are ultimately pseudo-periodic (UPP) represent good models for both traffic and service guarantees. Moreover, they prove that this class of functions is closed under the main NC operations and can be described with a finite amount of information. Additionally, they introduce the algorithms that embody the main NC operations, computing UPP results starting from UPP operands. The results in these works cover the main operations used in NC, such as minimum, min-plus convolution, min-plus deconvolution, etc. (see [20] for a complete list). The toolbox was first implemented in the COINC free library [21], which is not available anymore, and later by the commercial library RTaW-Pegase [22].

However, other NC operators, i.e., the composition and lower and upper pseudo-inverse, have been the focus of recent NC literature. In [20, Theorem 8.6], lower pseudo-inverse and composition are used to compute

[1] We use the terms *function* and *curve* interchangeably.

the per-flow service curve for a Weighted Round-Robin scheduler; in [23, Theorem 1], a similar result is shown for an Interleaved Weighted Round-Robin scheduler, using again lower pseudo-inverse and composition; in [24], authors show that several service curves can be found for a flow scheduled in a Deficit Round-Robin scheduler, under different hypotheses regarding cross-traffic. Works [25, 26] use pseudo-inverses and compositions to study properties of IEEE Time-Sensitive Networking (TSN) [27], a standard relevant for many applications. Work [28] shows the duality between min-plus and max-plus models, and how the lower and upper pseudo-inverses can be used to switch between the two models. This is exploited in [29] to devise an alternative algorithm for min-plus convolution, which transforms it into a max-plus convolution, obtaining a considerable speedup in the settings discussed in that paper. We can therefore obtain the results presented in these papers, using arbitrarily complex UPP curves as inputs.

While the algebraic formulation of these three operations is well known, their algorithmic aspects have not been addressed, to the best of our knowledge. This means that we do not have publicly known algorithms that compute these operations yet. In this paper, we aim to fill this gap and extend the existing algorithm toolbox to include lower- and upper-pseudo inverses and composition of functions.

We show that the UPP class is closed with respect to these operations, and provide algorithms to compute the result of each one. We prove that all of them have linear complexity with respect to the number of segments that represent the operands. We design specialized, more efficient versions of the composition algorithm that leverage characteristics of the operand functions – notably, their being Ultimately Affine (UA) or Ultimately Constant (UC) [20]. Last, we exemplify our findings on a comprehensive proof of concept, showing how to compute the per-flow service curve of [23, Theorem 1]. The algorithms described in this paper, together with those for known NC operators, are implemented in the Nancy open-source toolbox [30], which, to the best of our knowledge, is the only public one to implement UPP algorithms.

The rest of this paper is organized as follows: Section 2 briefly introduces NC notation and some basic results. Section 3 introduces the definitions and notation used throughout the paper, and discusses the kind of results that we need to provide for each operator to enable their implementation. In Section 4, we present the results for the lower and upper pseudo-inverse operators, including their properties for UPP curves and algorithms to compute them. Section 5 shows our results for the composition operator, including its properties on UPP curves and an algorithm to compute it. In Section 6, we report a proof-of-concept evaluation, by computing the results of a recent NC paper via our algorithms. Finally, Section 7 draws some conclusions and highlights directions for future works.

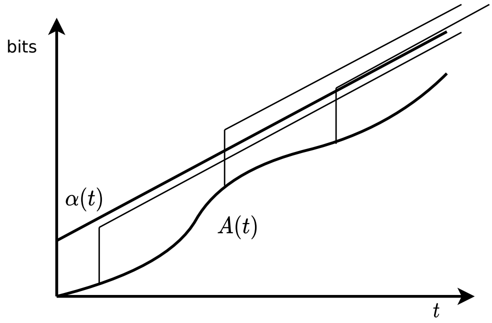


Fig. 1: Example of leaky-bucket shaper, taken from [17]. The traffic process $A(t)$ is always below the arrival curve $\alpha(t)$ and its translations along $A(t)$.

2 Network Calculus Basics

This section briefly introduces Network Calculus (NC). We use here the same notation as in [3], to which the interested reader is referred for a more in-depth explanation. A NC flow is represented by a function of time $A(t)$ that counts the amount of traffic arrived by time t. Such function is necessarily non-decreasing. It is often assumed to be left-continuous, i.e., $A(t)$ represents the number of bits in $[0, t[$. In particular, $A(0) = 0$.

Flows can be constrained by *arrival curves*. A non-decreasing function α is an *arrival curve* for a flow A if

$$A(t) - A(s) \leq \alpha(t - s), \qquad \forall s \leq t.$$

For instance, a *leaky-bucket shaper*, with a *rate* ρ and a *burst size* σ, enforces the affine arrival curve

$$\gamma_{\sigma,\rho}(t) = \begin{cases} \sigma + \rho t, & \text{if } t > 0, \\ 0, & \text{otherwise,} \end{cases}$$

as shown in Figure 1. In particular, this means that the long-term arrival rate of the flow cannot exceed ρ. Leaky-bucket shapers are often employed at the entrance of a network, to ensure that the injected traffic does not exceed the negotiated amount.

Let A and D be non-decreasing functions that describe the same data flow at the input and output of a lossless network element (or *node*), respectively. [2] If that node does not create data internally (which is often the case), causality requires that $A \geq D$. We say that the node guarantees to the flow a (minimum) *service curve* β if

$$D(t) \geq \inf_{0 \leq s \leq t} \{A(s) + \beta(t - s)\} =: (A \otimes \beta)(t), \qquad \forall t \geq 0. \tag{1}$$

[2] The function argument t is omitted whenever it is clear from the context.

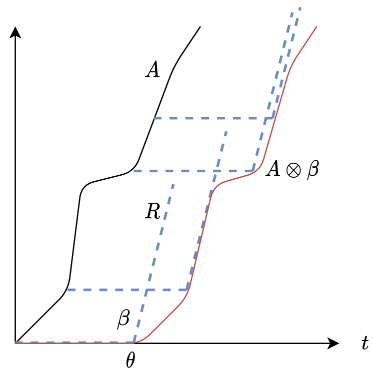


Fig. 2: Graphical interpretation of the convolution operation. A is the input function, $\beta_{R,\theta}$ is a rate-latency service curve, and $A \otimes \beta$ is a lower bound on the output.

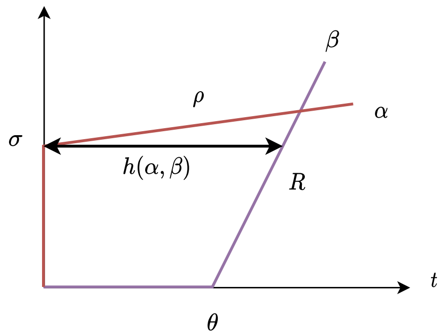


Fig. 3: Graphical example of a delay bound.

We call the operation on the right the *min-plus convolution* of A and β. Several network elements, such as delay elements, schedulers or links, can be modeled through service curves.

A very frequent case is the one of *rate-latency* service curves, defined as

$$\beta_{R,\theta}(t) = R \cdot [t - \theta]^+$$

for some rate $R > 0$ and latency $\theta \geq 0$. We write $[\cdot]^+$ to denote $\max\{\cdot, 0\}$. For instance, a constant-rate server (e.g., a wired link) can be modeled as a rate-latency service curve with zero latency. Figure 2 shows the lower bound of D obtained by computing $A \otimes \beta$, with $\beta = \beta_{R,\theta}$.

A point of strength of NC is that service curves are *composable*: the end-to-end service curve of a tandem of nodes traversed by the same flow can be computed as the min-plus convolution of the service curves of each node.

For a flow that traverses a service curve (be it the one of a single node, or the end-to-end service curve of a tandem computed as discussed above), an *upper bound* on the delay can be computed by combining its arrival curve α and the service curve β itself, as follows:

$$h(\alpha, \beta) = \sup_{t \geq 0} \left\{ \inf \left\{ d \geq 0 \mid \alpha(t - d) \leq \beta(t) \right\} \right\}. \tag{2}$$

The quantity $h(\alpha, \beta)$ is in fact the maximum horizontal distance between α and β, as shown in Figure 3. Therefore, computing the end-to-end service curve of a flow in a tandem traversal is the crucial step towards obtaining its worst-case delay bound.

The above introduction, albeit concise, should convince the alert reader that algorithms for automated manipulation of curves, implementing NC operators, are necessary to reap the benefits of NC algebra in practical scenarios. Many such algorithms have been discussed in [19, 20]. The next section describes the generic algorithmic framework exposed in these papers, which we extend in this work.

3 Mathematical Background and Notation

In this section, we provide an overview of the mathematical background for this paper, including the definitions used and the results we aim to provide.

NC computations can be implemented in software. In order to do so, one needs to provide finite representations of functions and well-formed algorithms for NC operations. According to the widely accepted approach described in [19, 20], a sufficiently generic class of functions useful for NC computations is the set $\mathcal{U}$ of (i) ultimately pseudo-periodic (ii) piecewise affine $\mathbb{Q}_+ \to \mathbb{Q} \cup \{+\infty, -\infty\}$ functions. We define both properties (i) and (ii) separately:

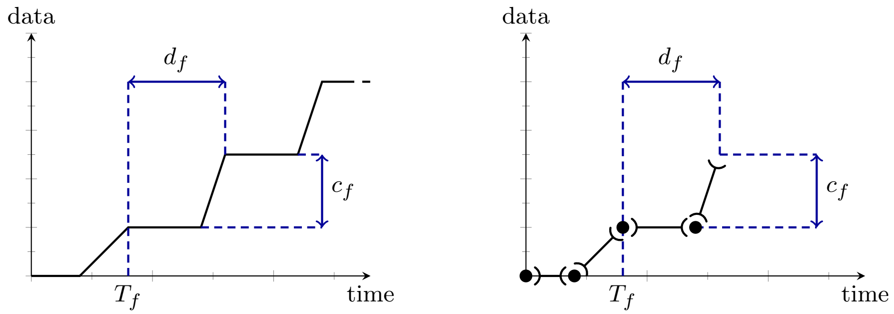
Definition 1 (Ultimately Pseudo-Periodic Function [19, p. 8]) Let f be a function $\mathbb{Q}_+ \to \mathbb{Q} \cup \{+\infty, -\infty\}$. Then, f is ultimately pseudo-periodic (UPP) if there exist $T_f \in \mathbb{Q}_+, d_f \in \mathbb{Q}_+ \setminus \{0\}, c_f \in \mathbb{Q} \cup \{+\infty, -\infty\}$ such that[3]

$$f(t + k \cdot d_f) = f(t) + k \cdot c_f, \qquad \forall t \geq T_f, \forall k \in \mathbb{N}. \tag{3}$$

We call T_f the *(pseudo-periodic) start* or length of the initial transient, d_f the *(pseudo-periodic) length*, and c_f the *(pseudo-periodic) height*.

Definition 2 (Piecewise Affine Function [19, p. 9]) We say that a function f is *piecewise affine* (PA) if there exists an increasing sequence $(a_i), i \in \mathbb{N}_0$ which tends to $+\infty$, such that $a_0 = 0$ and $\forall i \in \mathbb{N}_0$, it either holds that $f(t) = b_i + \rho_i t$ for some $b_i, \rho_i \in \mathbb{Q}$, or $f(t) = +\infty$, or $f(t) = -\infty$ for all $t \in \left]a_i, a_{i+1}\right[$. The a_i's are called *breakpoints*.

[3] We denote the set of non-negative numbers $\{0, 1, 2, 3, \ldots\}$ by $\mathbb{N}_0$ and the set of strictly positive numbers $\{1, 2, 3, \ldots\}$ by $\mathbb{N}$.



(a) f (b) R_f

Fig. 4: Example of ultimately pseudo-periodic piecewise affine function f and its representation R_f.

In [19], this class of functions is shown to be stable w.r.t. all min-plus operations, while functions $\mathbb{R}_+ \to \mathbb{R} \cup \{+\infty, -\infty\}$ are not.[4] We remark that functions in $\mathcal{U}$ are not necessarily wide-sense increasing. While NC functions are usually assumed to be so, in order to implement min-plus operations it is sometimes useful to include non-monotonic functions as well. Similarly, functions in $\mathcal{U}$ can assume infinite values. This is also useful for algebraic manipulations, e.g., to express a function as a minimum of two or more functions. Throughout this paper, we will consider all functions to be in $\mathcal{U}$, hence, piecewise affine and UPP. For such functions, it is enough to store a representation of the initial transient part and of one period, which is a finite amount of information. This is exemplified in Figure 4.

Accordingly, we call a *representation* R_f of a function f the tuple (S, T, d, c), where T, d, c are the values described above, and S is a sequence of points and open segments describing f in $[0, T + d[$. We use both points and open segments in order to easily model discontinuities. We will use the umbrella term *elements* to encompass both when convenient.

Definition 3 (Point) We define a *point* as a tuple

$$p_i \coloneqq (t_i, f(t_i)), \qquad i \in \{1, \dots, n\}.$$

Definition 4 (Segment) We define a *segment* as a tuple

$$s_i \coloneqq \left(t_i, t_{i+1}, f(t_i^+), f(t_{i+1}^-)\right), \qquad i \in \{1, \dots, n\}$$

which describes f in the open interval $]t_i, t_{i+1}[$ in which it is affine, i.e.,

$$f(t) = f(t_i^+) + \frac{f(t_{i+1}^-) - f(t_i^+)}{t_{i+1} - t_i} \cdot (t - t_i) =: b + r \cdot (t - t_i) \qquad \text{for all } t \in]t_i, t_{i+1}[,$$

[4] An alternative class of functions with such stability is $\mathbb{N} \to \mathbb{R}$, however this is only feasible for models where time is discrete.

where we used the following shorthand notation for one-sided limits:

$$f\left(t_i^+\right) = \lim_{t \to t_i^+} f(t)\,,\ f\left(t_i^-\right) = \lim_{t \to t_i^-} f(t)\,.$$

If $r = 0$, we call s_i a *constant segment*.

Definition 5 (Sequence) We define a sequence S_f^D as on ordered set of elements $e_1, \ldots, e_n$ that alternate between points and segments and describe f in the finite interval $D \subset \mathbb{Q}_+$.

For example, given $D = [0, T[$, then $S_f^D = \{p_1, s_1, p_2, \ldots, p_n, s_n\}$ where $p_1 = (0, f(0))$ and, assuming $p_n = (t_n, f(t_n))$ for some $0 < t_n < T$, $s_n = (t_n, T, f(t_n^+), f(T^-))$.

Note that, given R_f, one can compute $f(t)$ for all $t \geq 0$, and also S_f^D for any interval D. Furthermore, being finite, R_f can be used as a data structure to represent f in code. As discussed in depth in [31], R_f is not unique, and using a *non-minimal* representation of f can affect the efficiency of the computations. Work [31] also describes an efficient algorithm that minimizes a representation R_f (i.e., computes the smallest T_f, d_f that are required to represent f). Given a sequence S, let $n(S)$ be its cardinality. As it is useful in the following, we define *Cut* to be an (obvious) algorithm that, given R_f and D, computes S_f^D. With a little abuse of notation, we use min-plus operators directly on finite sequences such as S_f^D. For instance, given the lower pseudo-inverse of f, $(f)_\downarrow^{-1}$ (its formal definition is in the next section), we will write $\left(S_f^D\right)_\downarrow^{-1}$, to express that we are computing it on f over the limited interval D.

A NC operator can then be defined computationally as an algorithm that takes UPP representations of its input functions and yields a UPP representation of the result, provided that the class of UPP functions is closed with respect to such operator. Considering a generic unitary operator $[\cdot]^*$, in order to compute f^* we need an algorithm that computes R_{f^*} from R_f, i.e., $R_f \to R_{f^*}$. We call this *by-curve* algorithm. This process can be divided in the following steps:

1. compute valid parameters T_{f^*}, d_{f^*} and c_{f^*} for the result;
2. compute $S_f^D \to S_{f^*}$, i.e., use an algorithm that computes the resulting sequence from the sequence of the operand. We call this *by-sequence* algorithm for operator $[\cdot]^*$. In order to run this algorithm, a suitable domain D must be identified, based on the properties of operator $[\cdot]^*$, and, accordingly, one must compute sequence $S_f^D = \mathrm{Cut}(R_f, D)$;
3. return $R_{f^*} = (S_{f^*}, T_{f^*}, d_{f^*}, c_{f^*})$.

We therefore need to provide the following results:

- a proof that the result of the operator $[\cdot]^*$, applied to a UPP function, yields a UPP result;
- a way to compute UPP parameters T_{f^*}, d_{f^*} and c_{f^*} from R_f;
- a valid domain D, again to be computed from R_f;
- a *by-sequence* algorithm.

Combining the above results, we can then construct the *by-curve* algorithm for operator $[\cdot]^*$, which allows one to compute $[\cdot]^*$ for any UPP curve.[5]

We exemplify the above by showing the by-curve algorithm for the *left shift* of a function by $\tau \geq 0$. Given $f(t)$, whose representation is $R_f = (S_f, T_f, d_f, c_f)$, we want to compute the representation $R_g = (S_g, T_g, d_g, c_g)$ of $g(t) = f(t + \tau)$. Quite intuitively, it is $T_g = [T_f - \tau]^+$, $d_g = d_f$, $c_g = c_f$. Moreover, the valid domain D where we need to define sequence S_f^D for the by-sequence algorithm is $D = [\tau, \max\{\tau, T_f\} + d_f[$. We leave to the interested reader the straightforward (yet tedious) task of deriving sequence S_g from S_f^D, i.e., of figuring out the by-sequence algorithm for this operator.

Works [19, 20] provided such computational descriptions for fundamental NC operators such as minimum, sum, convolution, and many others. In this paper, we extend the above toolbox by adding the lower pseudo-inverse, upper pseudo-inverse, and composition operators. To the best of our knowledge, no computational description of the above has been formalized before, despite their relevance in the NC literature.

Before presenting our contribution, we introduce two more definitions that will be used throughout the paper.

Definition 6 (Ultimately Affine Function) Let f be a function $\mathbb{Q}_+ \to \mathbb{Q} \cup \{+\infty, -\infty\}$. Then, f is Ultimately Affine (UA), if either there exist $T_f^a \in \mathbb{Q}_+, \rho_f \in \mathbb{Q}$ such that

$$f(t) = f(T_f^a) + \rho_f \cdot \left(t - T_f^a\right), \qquad \forall t \geq T_f^a, \tag{4}$$

or $f(t) = +\infty$, or $f(t) = -\infty$ for all $t \geq T_f^a$.

Note that this definition differs from the one in the literature [19], but we prove their equivalence in Appendix A. UA functions are (obviously) UPP as well, their period being a single segment of slope ρ_f and arbitrary length starting at T_f^a. They occur quite often in NC, e.g., the arrival curve of a leaky-bucket shaper or a rate-latency service curve are both UA. An *Ultimately Constant* (UC) function is UA with $\rho_f = 0$. Similarly, an *Ultimately Infinite* (UI) function is UA with $f(t) = +\infty$, or $f(t) = -\infty$ for all $t \geq T_f^a$.

Unlike UPP, the class of UA functions is not closed with respect to NC operations. For instance, [31] shows that flow-controlled networks with rate-latency (hence UA) service curves yield closed-loop service curves that are UPP, but not necessarily UA again. Moreover, in many cases, the service curves of individual flows served by Round-Robin schedulers are UPP, but not UA either (see, e.g., [23, 24, 32]). However, there are cases when simpler algorithms for NC operations can be derived if one assumes that operands are UA. For this reason, there are NC toolboxes that only consider UA functions, e.g., [33]. A possible approach to NC analysis is thus to approximate UPP (non-UA) functions with UA lower/upper bounds, trading some accuracy for computation time [34, 35]. Throughout this paper, we provide general

[5] The same process applies also, with minor adjustments, to binary operators.

algorithms for UPP functions. However, we also show what is to be gained – in terms of domain compactness and/or algorithmic efficiency – when we can make stronger assumptions on the operands.

4 Lower and Upper Pseudo-Inverse of UPP Functions

In this section, we discuss the lower and upper pseudo-inverse operators for UPP functions. Henceforth, we will omit the *lower* or *upper* attribute when the discussion applies to both.

First, we provide formal definitions.

Definition 7 (Lower and Upper Pseudo-Inverse) Let $f \in \mathcal{U}$ be non-decreasing. Then its *lower pseudo-inverse* is defined as

$$f_{\downarrow}^{-1}(y) := \inf \{t \geq 0 \mid f(t) \geq y\},$$

and its *upper pseudo-inverse* is defined as

$$f_{\uparrow}^{-1}(y) := \sup \{t \geq 0 \mid f(t) \leq y\}.$$

We can find an equivalent definition as follows.

Proposition 8 *Let $f \in \mathcal{U}$ be non-decreasing. For all $y > f(0)$, its lower pseudo-inverse is equal to*

$$f_{\downarrow}^{-1}(y) = \sup \{t \geq 0 \mid f(t) < y\}, \tag{5}$$

and for all $y \geq f(0)$, its upper pseudo-inverse is equal to

$$f_{\uparrow}^{-1}(y) = \inf \{t \geq 0 \mid f(t) > y\}. \tag{6}$$

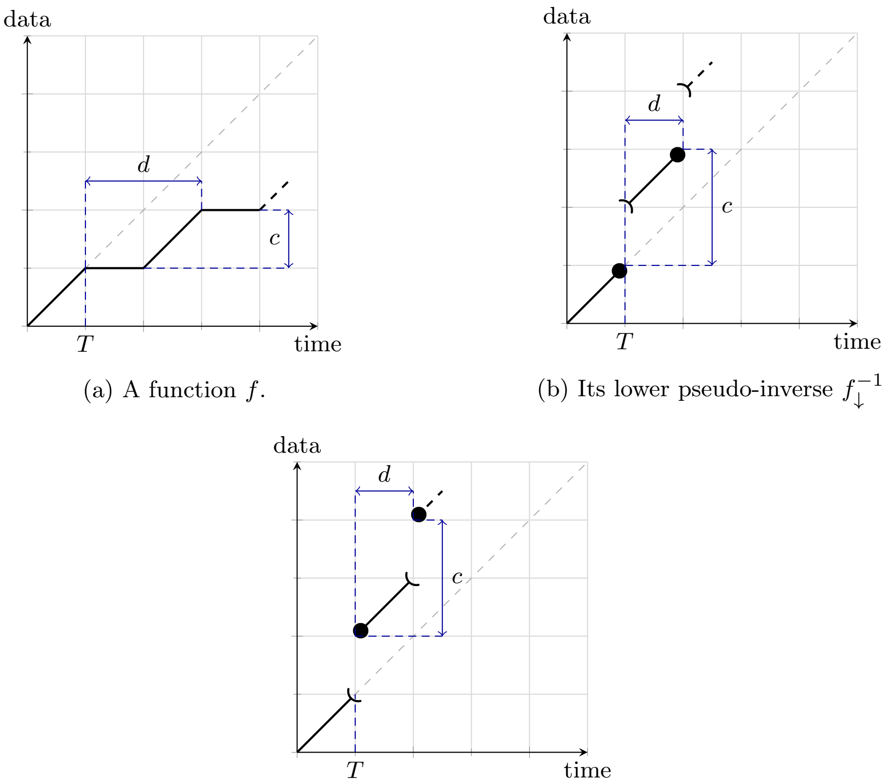
Note that [28] reports a slightly different definition, because functions are defined in $\mathbb{R} \to \mathbb{R}$. Our functions in $\mathcal{U}$ are defined in $\mathbb{Q}_+ \to \mathbb{Q}$, hence our domain is lower bounded. The consequences of this difference are discussed in Appendix B, which also contains a proof of Proposition 8.

Note that the lower pseudo-inverse is left-continuous and the upper pseudo-inverse is right-continuous [28, p. 64]. Moreover, we have in general that [28, p. 61]

$$f_{\downarrow}^{-1} \leq f_{\uparrow}^{-1}.$$

An example of these operators is shown in Figure 5.

Figure 5 shows a UPP function and its lower and upper pseudo-inverses. In NC, both pseudo-inverses are useful to switch from min-plus to max-plus algebra and vice versa [28]. Later on, in Section 5, we provide the examples of Equation (15) which uses the lower pseudo-inverse in conjunction with the composition operator, and of Algorithm 3 which shows that the lower pseudo-inverse is required to compute the composition between two UPP curves.



(a) A function f.

(b) Its lower pseudo-inverse $f_{\downarrow}^{-1}$.

(c) Its upper pseudo-inverse $f_{\uparrow}^{-1}$.

Fig. 5: Example of lower and upper pseudo-inverse of a function f.

The rest of this section is organized as follows. In Section 4.1 we show that the pseudo-inverse of a UPP function is still UPP, and provide expressions to compute its UPP parameters a priori. In Section 4.2 we discuss, first through a visual example and then via pseudocode, how to algorithmically compute the pseudo-inverse. In Section 4.3 we conclude with a summary of the by-curve algorithm, some observations on the algorithmic complexity of this operator. In Section 4.4 we discuss corner cases.

4.1 Properties of pseudo-inverses of UPP functions

We discuss our properties for a generic function f, *excluding* the cases of UC and UI functions. These two cases are treated separately for ease of presentation. At the end of this section, we report the necessary information for the alert reader to retrace the steps exposed hereafter to include these two corner cases. We remark that the Nancy software library [30] computes pseudo-inverses of generic (non-decreasing) UPP functions, including UC and UI ones.

Theorem 9 *Let $f \in \mathcal{U}$ be a non-decreasing function that is neither UC nor UI. Then, its lower pseudo-inverse $f_{\downarrow}^{-1}(x) = \inf \{t \mid f(t) \geq x\}$ is again a function $\in \mathcal{U}$ with*

$$T_{f_{\downarrow}^{-1}} = f\left(T_f + d_f\right), \tag{7}$$

$$d_{f_{\downarrow}^{-1}} = c_f, \tag{8}$$

$$c_{f_{\downarrow}^{-1}} = d_f. \tag{9}$$

Proof Let $t_1 \geq T_f + d_f$ and $x := f(t_1)$. Moreover, we define

$$t_0 := f_{\downarrow}^{-1}(x) = \inf \{t \mid f(t) \geq x\} = \inf \{t \mid f(t) \geq f(t_1)\}.$$

By definition, it is clear that $t_0 \leq t_1$ (t_1 satisfies the condition inside the infimum, and t_0 is its largest lower bound). Moreover, since it holds that $f(t+d_f) = f(t)+c_f$ for all $t \geq T_f$, we can conclude that, for all $\tau \geq T_f + d_f$,

$$f(\tau) = f\left((\tau - d_f) + d_f\right) = f(\tau - d_f) + c_f.$$

Thus,

$$f(\tau - d_f) = f(\tau) - c_f. \tag{10}$$

Since f is non-UC (i.e., $c_f > 0$), and we have by definition $t_1 \geq T_f + d_f$, it follows that

$$f(T_f) \leq f(t_1 - d_f) \overset{(10)}{=} f(t_1) - c_f < f(t_1) = f(t_0),$$

where we used in the strict inequality that f is not UC and thus $t_0 > T_f$. Therefore, for any $k \in \mathbb{N}$,

$$\begin{aligned} f_{\downarrow}^{-1}\left(x + k \cdot d_{f_{\downarrow}^{-1}}\right) &= \inf\left\{t \mid f(t) \geq x + k \cdot d_{f_{\downarrow}^{-1}}\right\} \\ &\overset{(8)}{=} \inf\left\{t \mid f(t) \geq x + k \cdot c_f\right\} \\ &= \inf\left\{t \mid f(t) \geq f(t_1) + k \cdot c_f\right\} \\ &= \inf\left\{t \mid f(t) \geq f(t_0) + k \cdot c_f\right\} \\ &= \inf\left\{t \mid f(t) \geq f(t_0 + k \cdot d_f)\right\} \\ &= t_0 + k \cdot d_f \\ &\overset{(9)}{=} f_{\downarrow}^{-1}(x) + k \cdot c_{f_{\downarrow}^{-1}}. \end{aligned}$$

□

It follows from Theorem 9 that, in order to compute a representation $R_{f_{\downarrow}^{-1}}$, we need only to compute $S_{f_{\downarrow}^{-1}}^{D'}$ where

$$D' = \left[0, T_{f_{\downarrow}^{-1}} + d_{f_{\downarrow}^{-1}}\right[= [0, f(T_f + d_f) + c_f[.$$

If there is no left-discontinuity in $T_f + 2 \cdot d_f$, it follows that

$$S^{D'}_{f_{\downarrow}^{-1}} = \left(S^D_f\right)^{-1}_{\downarrow},$$

where

$$D = [0, T_f + 2 \cdot d_f[\,. \tag{11}$$

Otherwise, let $x_1 = f\left((T_f + 2 \cdot d_f)^-\right)$ and $x_2 = f\left(T_f + 2 \cdot d_f\right)$, then $x_1 < x_2$, and therefore $S^{D'}_{f_{\downarrow}^{-1}}$ must end with a constant segment defined in $]x_1, x_2[$ with ordinate $T_f + 2 \cdot d_f$. Such segment must be added manually at the end of $\left(S^D_f\right)^{-1}_{\downarrow}$.

A similar result can be derived for the upper pseudo-inverse.

Theorem 10 *Let $f \in \mathcal{U}$ be a non-decreasing function that is neither UC nor UI. Then, the upper pseudo-inverse $f_{\uparrow}^{-1}(x) = \sup\{t \mid f(t) \le x\}$ is again a function $\in \mathcal{U}$ with*

$$T_{f_{\uparrow}^{-1}} = f\left(T_f\right), \tag{12}$$

$$d_{f_{\uparrow}^{-1}} = c_f, \tag{13}$$

$$c_{f_{\uparrow}^{-1}} = d_f. \tag{14}$$

Proof The proof follows the same steps as the one for the lower pseudo-inverse. Let $t_0 \ge T_f$ and $x \coloneqq f(t_0)$. Moreover, we define

$$t_1 \coloneqq f_{\uparrow}^{-1}(x) = \sup\{t \mid f(t) \le x\} = \sup\{t \mid f(t) \le f(t_0)\}.$$

By definition, it is clear that $t_0 \le t_1$ (t_0 satisfies the condition in the supremum, and t_1 is the largest to satisfy it). Since f is non-UC, and we have by definition $t_0 \ge T_f$, it follows that

$$f(t_0 + d_f) \overset{(3)}{=} f(t_0) + c_f > f(t_0) = f(t_1),$$

where we used in the strict inequality that f is not ultimately constant and thus $t_1 < t_0 + d_f < \infty$. Therefore, for any $k \in \mathbb{N}$,

$$\begin{aligned}
f_{\uparrow}^{-1}\left(x + k \cdot d_{f_{\uparrow}^{-1}}\right) &= \sup\left\{t \mid f(t) \le x + k \cdot d_{f_{\uparrow}^{-1}}\right\} \\
&\overset{(13)}{=} \sup\left\{t \mid f(t) \le x + k \cdot c_f\right\} \\
&= \sup\left\{t \mid f(t) \le f(t_0) + k \cdot c_f\right\} \\
&= \sup\left\{t \mid f(t) \le f(t_1) + k \cdot c_f\right\} \\
&= \sup\left\{t \mid f(t) \le f(t_1 + k \cdot d_f)\right\} \\
&= t_1 + k \cdot d_f \\
&\overset{(14)}{=} f_{\uparrow}^{-1}(x) + k \cdot c_{f_{\uparrow}^{-1}}.
\end{aligned}$$

□

Similar to the previous theorem, it follows from Theorem 10 that, in order to compute a representation $R_{f_\uparrow^{-1}}$, we need only to compute $S_{f_\uparrow^{-1}}^{D'}$, where

$$D' = \left[0, T_{f_\uparrow^{-1}} + d_{f_\uparrow^{-1}}\right[= [0, f(T_f) + c_f[\,.$$

If there is no left-discontinuity in $T_f + d_f$, it follows that

$$S_{f_\uparrow^{-1}}^{D'} = \left(S_f^D\right)_\uparrow^{-1},$$

where

$$D = [0, T_f + d_f[\,.$$

Otherwise, let $x_1 = f\left((T_f + d_f)^-\right)$ and $x_2 = f\left(T_f + d_f\right)$, then $x_1 < x_2$, and therefore $S_{f_\uparrow^{-1}}^{D'}$ must end with a constant segment defined in $]x_1, x_2[$ with ordinate $T_f + d_f$. Such segment must be added manually at the end of $\left(S_f^D\right)_\uparrow^{-1}$.
The alert reader will notice that $T_{f_\downarrow^{-1}}$ and $T_{f_\uparrow^{-1}}$ differ, for which we can provide the following intuitive explanation. Consider a function f so that $f(t) = k, \forall t \in]a, T+b[$ with $a < T, b > 0$. Then $f_\downarrow^{-1}(k) = a$, as the lower pseudo-inverse "goes backwards" to the start of the constant segment. However, since $a < T$, the pseudo-periodic property does not apply for $f(a)$, i.e., we cannot say anything about $f(a+d)$. So, in general, we cannot say $f_\downarrow^{-1}$ is pseudo-periodic from $f(T)$, and we instead need to "skip" to the second pseudo-period so that, as in the proof, $T < a < T + d$.
The same does not apply for $f_\uparrow^{-1}$, however, since $f_\uparrow^{-1}(k) = T + b$ as the upper pseudo-inverse "goes forward" to the end of the constant segment and $T + b > T$, thus we can rely on the pseudo-periodic property of f.

An interesting consequence of this discussion is that the representation R_f may change when we do not expect it to. From [28, p. 64], [20, p. 48], we know the following properties:

- if f is left-continuous, $\left(f_\downarrow^{-1}\right)_\downarrow^{-1} = \left(f_\uparrow^{-1}\right)_\downarrow^{-1} = f$,
- if f is right-continuous, $\left(f_\uparrow^{-1}\right)_\uparrow^{-1} = \left(f_\downarrow^{-1}\right)_\uparrow^{-1} = f$.

Thus, one may expect that applying the pseudo-inverse twice would lead to a function with the *same* representation, i.e., that $\left((R_f)_\downarrow^{-1}\right)_\downarrow^{-1} = R_f$. However, as per the discussion above, the start of the pseudo-period of the result would wove from an initial T_f to $T_f + d_f + c_f$. This is unavoidable – the above example shows that there exists one case when T_f would *not* be the correct starting point. However, in other cases, T_f would be the correct starting point for the pseudo-periodic behavior.

This is an instance of a general issue encountered with algorithms for UPP curves – also discussed in [31]. Generic algorithms, that make no assumptions on the shape of the operands (such as the ones presented here for the pseudo-inverse), may in general yield *non-minimal representations* of the result. Generally speaking, minimal representations are preferable, since the number of elements in a sequence affects the complexity of the algorithms. However, addressing the issue of representation minimization *a priori* when implementing NC operators is too hard (if doable at all), since one would need to make a comprehensive list of subcases, and, of course, as many formal proofs of correctness. It is instead considerably more efficient to devise a generic algorithm for an operator, neglecting minimization, and then use a simple algorithm *a posteriori* that minimizes the representation of the result – see [31].

4.2 By-sequence algorithm for pseudo-inverses

In this section we discuss the by-sequence algorithms for pseudo-inverses. We recall that with "by-sequence" we mean that the operand, and thus its result, is defined on a limited domain. Without loss of generality, we will focus on a sequence S representing a function f over an interval $[0, t[$, with $f(0) = 0$. Then, $S_{\downarrow}^{-1}$ is the sequence representing $f_{\downarrow}^{-1}$ over interval $[0, f(t^-)[$. The same applies to $S_{\uparrow}^{-1}$.

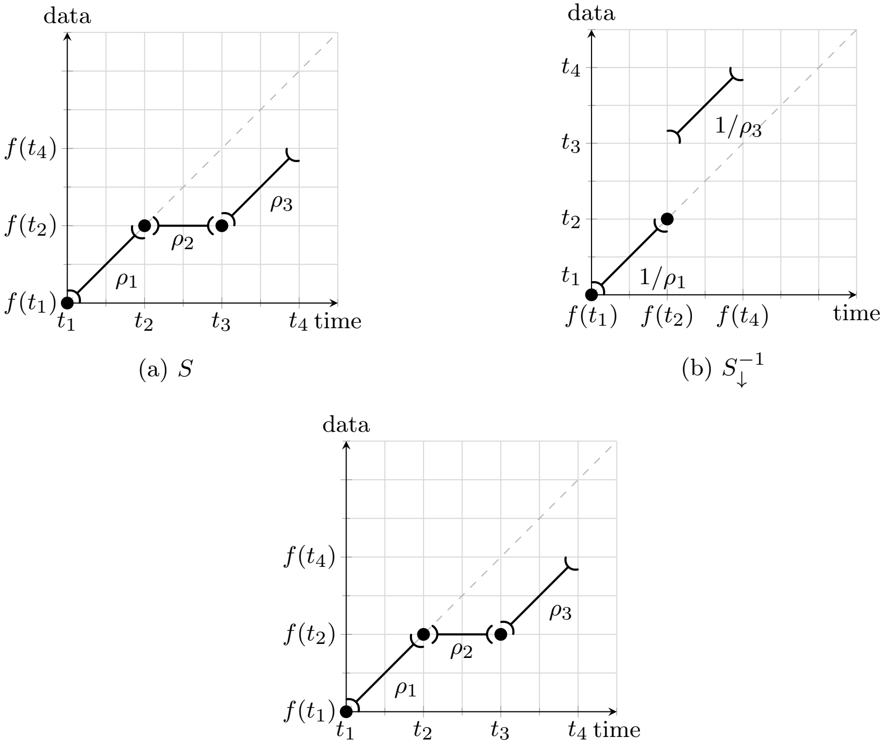
The simplest case is when S is continuous and strictly increasing, hence bijective. In this case, both $S_{\downarrow}^{-1}$ and $S_{\uparrow}^{-1}$ are the classic *inverse* of S, and the algorithm consists of drawing, for each point and segment of S, its reflection over the line $y = x$. However, when S includes discontinuities and/or constant segments, the algorithm becomes slightly more complicated: a discontinuity in S "maps" to a constant segment in both $S_{\downarrow}^{-1}$ and $S_{\uparrow}^{-1}$, while a constant segment in S "maps" to a right-discontinuity in $S_{\downarrow}^{-1}$ and a left-discontinuity in $S_{\uparrow}^{-1}$. This is exemplified in Figure 6.

We describe Algorithm 1 for the lower pseudo-inverse (the one for the upper pseudo-inverse differs in few details which we briefly discuss later). Algorithm 1 linearly scans S considering one element at a time. Based on the type of element (point or segment), as well as on its topological relationship with its predecessor, it decides what to add to $S_{\downarrow}^{-1}$.

More in detail, there are eight possible cases, shown in Table 1, which require zero, one, two, or three elements to be added to $S_{\downarrow}^{-1}$. These are reported in the same order in Algorithm 1. The rigorous (though cumbersome) mathematical justification for each case is instead postponed to Appendix C for the benefit of the interested reader.

We exemplify the above algorithm with reference to the example in Figure 6. For each of the considered steps, we will reference the case in Table 1, the line of Algorithm 1, and the relevant equations from Appendix C proving the result. Processing each element from left to right, we calculate:

- The origin $(t_1, f(t_1)) = (0, 0)$ for $f_{\downarrow}^{-1}(0)$.
- For the segment s_1 and its predecessor point $p_1 = (t_1, f(t_1))$: this corresponds to Line 22 of the algorithm. Since s_1 has a positive slope, we continue in Line 31. As the function is right-continuous at t_1, we are in



(a) S

(b) $S_{\downarrow}^{-1}$

(c) $\left(S_{\downarrow}^{-1}\right)_{\downarrow}^{-1}$

Fig. 6: Example of lower pseudo-inverse of a sequence S. Since S is left-continuous, $S = \left(S_{\downarrow}^{-1}\right)_{\downarrow}^{-1}$.

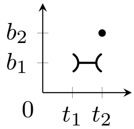
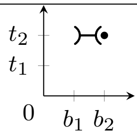
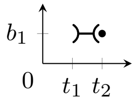
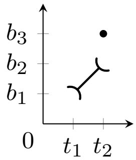
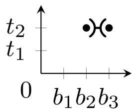
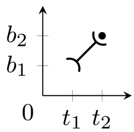
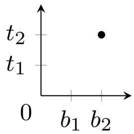
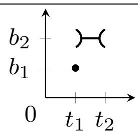
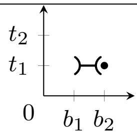
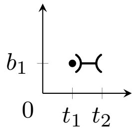
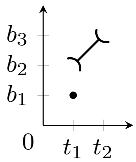
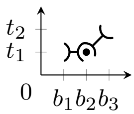
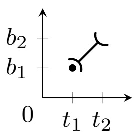
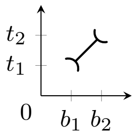
case c8. We go to Line 36 and add a segment $s = \left(f(t_1^+), f(t_2^-), t_1, t_2\right)$ to O. It can be verified that this follows Equation (43).

- For the point $p_2 = (t_2, f(t_2))$ and its preceding segment s_1, we are in case c4, corresponding to Line 18 and we therefore append the point $p := (f(t_2), t_2)$ to O. It can be verified that this follows Equation (35).
- For the constant segment s_2 with preceding point $p_2 = (t_2, f(t_2))$, we are in case c6, corresponding to Line 28, and no element is added. This follows Equation (39).
- For the point $p_3 = (t_3, f(t_3))$ with preceding constant segment s_2, we are in case c2, corresponding to Line 10, and no element is added. This follows Equation (31).
- For a segment s_3 with preceding point $(t_3, f(t_3))$, we are in case c8, Line 36, and append $s := \left(f(t_3^+), f(t_4^-), t_3, t_4\right)$ to O (verified in Equation (43)).

We note that, since $S_{\downarrow}^{-1}$ is left-continuous, when a continuous sequence of a point, a constant segment, and a point is encountered in S, they all "map" to the inverse of the first (leftmost) point of this sequence. This justifies the fact that nothing has to be added to $S_{\downarrow}^{-1}$ in these cases (e.g., 2 and 6).

The algorithm for $S_{\uparrow}^{-1}$, that we omit here for brevity, differs from the one provided only in how constant segments are handled, that is, by appending the inverse of the *last* (rightmost) point instead of the first (recall that the upper pseudo-inverse is right-continuous). This requires the algorithm for $S_{\uparrow}^{-1}$ to look ahead to the next element during the linear scan. We leave the (tedious, but simple) task of spelling out the minutiae of this algorithm to the interested reader.

Table 1: Cases to be considered in the by-sequence algorithm to compute $S_{\downarrow}^{-1}$

Considered Element	Constant segment	Discontinuity in S	Example of S	Append to $S_{\downarrow}^{-1}$	Case #
Point after segment	Yes	Yes			c1
		No		*nothing to append*	c2
	No	Yes			c3
		No			c4
Segment after point	Yes	Yes			c5
		No		*nothing to append*	c6
	No	Yes			c7
		No			c8

Algorithm 1 Pseudocode for lower pseudo-inverse of a finite sequence

Input Finite sequence of elements S, consisting of $e_k, k \in \{1, \ldots, n\}$ that is either a point or a segment. Moreover, e_1 is a point at the origin $(0, 0)$.

Return Lower pseudo-inverse $S_{\downarrow}^{-1}$ of S, consisting of a sequence of elements $O = \{o_1, \ldots, o_m\}$.

Define an empty sequence of elements $O \coloneqq \{\,\}$
Append $p \coloneqq (0, 0)$ to O ▷ $f_{\downarrow}^{-1}(e_0)$
for e_k **in** $(e_2, \ldots, e_n)$ **do** ▷ The for loop starts after the origin
    if $e_k == p_i$ **then** ▷ The element is a point
        e_{k-1} is a segment s_{i-1}
        if s_{i-1} is constant **then**
            if $f(t_i^-) < f(t_i)$ **then** ▷ f is not left-cont. at t_i
                Append $s \coloneqq \left(f(t_i^-), f(t_i), t_i, t_i\right)$ to O ▷ (c1)
                Append $p \coloneqq (f(t_i), t_i)$ to O
            else ▷ f is left-cont. at t_i
                Nothing to append ▷ (c2)
            end if
        else ▷ s_{i-1} is not constant
            if $f(t_i^-) < f(t_i)$ **then** ▷ f is not left-cont. at t_i
                Append $p \coloneqq \left(f(t_i^-), t_i\right)$ to O ▷ (c3)
                Append $s \coloneqq \left(f(t_i^-), f(t_i), t_i, t_i\right)$ to O
                Append $p \coloneqq (f(t_i), t_i)$ to O
            else ▷ f is left-cont. at t_i
                Append $p \coloneqq (f(t_i), t_i)$ to O ▷ (c4)
            end if
        end if
    else ▷ The element is a segment s_i
        e_{k-1} is a point p_i
        if $e_k = s_i$ is constant **then**
            if $f(t_i) < f(t_i^+)$ **then** ▷ f is not right-cont. at t_i
                Append $s \coloneqq \left(f(t_i), f(t_i^+), t_i, t_i\right)$ to O ▷ (c5)
                Append $p \coloneqq \left(f(t_i^+), t_i\right)$ to O
            else ▷ f is right-cont. at t_i
                Nothing to append ▷ (c6)
            end if
        else ▷ s_i is not constant
            if $f(t_i) < f(t_i^+)$ **then** ▷ f is not right-cont. at t_i
                Append $s \coloneqq \left(f(t_i), f(t_i^+), t_i, t_i\right)$ to O ▷ (c7)
                Append $p \coloneqq \left(f(t_i^+), t_i\right)$ to O
                Append $s \coloneqq \left(f(t_i^+), f(t_{i+1}^-), t_i, t_{i+1}\right)$ to O
            else ▷ f is right-cont. at t_i
                Append $s \coloneqq \left(f(t_i^+), f(t_{i+1}^-), t_i, t_{i+1}\right)$ to O ▷ (c8)
            end if
        end if
    end if
end for

4.3 By-curve algorithm for pseudo-inverses

We can now discuss the by-curve algorithm by combining the results presented in Sections 4.1 and 4.2. In Algorithm 2, we show the pseudocode to compute $f_{\downarrow}^{-1}$ for a UPP function f. The analogous for upper pseudo-inverse, which we omit for brevity, can be similarly derived from the results in the sections above.

Algorithm 2 Pseudocode for lower pseudo-inverse of a UPP function

---

Input Representation R_f of a non-decreasing UPP function f, consisting of sequence S_f and parameters T_f, d_f and c_f.

Return Representation $R_{f_{\downarrow}^{-1}}$ of $f_{\downarrow}^{-1}$.

Compute the UPP parameters for the result ▷ Theorem 9
  $T_{f_{\downarrow}^{-1}} \leftarrow f\left(T_f + d_f\right)$
  $d_{f_{\downarrow}^{-1}} \leftarrow c_f$
  $c_{f_{\downarrow}^{-1}} \leftarrow d_f$
Compute S_f^D ▷ Equation (11)
  $D \leftarrow [0, T_f + 2 \cdot d_f[$
  $S_f^D \leftarrow \mathrm{Cut}(R_f, D)$
Compute $S_{f_{\downarrow}^{-1}} \leftarrow \left(S_f^D\right)_{\downarrow}^{-1}$ ▷ Algorithm 1
$R_{f_{\downarrow}^{-1}} \leftarrow \left(S_{f_{\downarrow}^{-1}}, T_{f_{\downarrow}^{-1}}, d_{f_{\downarrow}^{-1}}, c_{f_{\downarrow}^{-1}}\right)$

Regarding the complexity of Algorithm 2, we note that the main cost is computing $\left(S_f^D\right)_{\downarrow}^{-1}$. Since Algorithm 1 is a linear scan of the input sequence, the resulting complexity is $\mathcal{O}\left(n\left(S_f^D\right)\right)$.

4.4 Corner cases: UC and UI functions

We conclude this section by discussing the two corner cases that we had initially left out, i.e., those when f is either Ultimately Constant (UC) or Ultimately Infinite (UI).

To obtain a representation of a UC or UI function, it is enough to find *any* T_f for which $f(t) = C$, $C \in \mathbb{Q}$, (UC) or $f(t) = +\infty$ (UI) for any $t \geq T_f$.[6] However, as we show in this section, the *infima* of the infinitely many points that verify the above play an important role in computing their pseudo-inverses. We provide formal definitions below:

Definition 11 Let $f \in \mathcal{U}$ be UC, and let $C \coloneqq \lim_{t \to +\infty} f(t)$, $C \in \mathbb{Q}$, be its (ultimately) constant value. Then, we define

$$T_C \coloneqq \inf\{T \mid f(t) = C,\ \forall t \geq T\}$$

---

[6]The definition of UI includes also $f(t) = -\infty$ for all $t \geq T_f$. However, since the pseudo-inverse operations only apply to non-decreasing functions, we do not consider such case here.

to be the infimum of its pseudo-periodic starts.

Note that we use an infimum, instead of a minimum, because f may not be right-continuous in T_C. In that case $f(t) = C$, $\forall t > T_C$, but $f(T_C) \neq C$.

Definition 12 Let $f \in \mathcal{U}$ be UI. Then, we define:

$$T_I := \inf\{T \mid f(t) = +\infty,\ \forall t \geq T\},$$

and

$$L = \begin{cases} f(T_I), & \text{if } f(T_I) < +\infty, \\ f(T_I^-), & \text{if } f(T_I) = +\infty \text{ and } T_I > 0, \\ 0, & \text{otherwise,} \end{cases}$$

i.e., L is the rightmost finite value of f.

Again, we use the infimum to include functions such that $f(t) = +\infty$, $\forall t > T_I$, but $f(T_I) = L < +\infty$.

As we assume in this section all functions to be non-decreasing, using Definition 11 we have that a UC function is such that

$$\begin{aligned} f(t) &< C, & \forall t &< T_C, \\ f(t) &= C, & \forall t &> T_C, \end{aligned}$$

whereas using Definition 12 a UI function is such that

$$\begin{aligned} f(t) &< +\infty, & \forall t &< T_I, \\ f(t) &= +\infty, & \forall t &> T_I. \end{aligned}$$

For these, some mathematical inconsistencies need be resolved first. For example:

- if f is UC, Algorithm 2 would yield $d_{f_\downarrow^{-1}} = 0$,
- if f is UI, it would yield $T_{f_\downarrow^{-1}} = +\infty$.

We treat these two cases in the following propositions.

Proposition 13 *Let $f \in \mathcal{U}$ be a non-decreasing, UC function with $T_C \in \mathbb{Q}_+$. If $f(T_C) < C$, its lower pseudo-inverse $f_\downarrow^{-1}(y)$ is*

$$f_\downarrow^{-1}(y) = \begin{cases} \inf\{x \mid f(x) \geq y\} = T_C, & \text{if } f(T_C) < y < C, \\ \inf\{x \mid f(x) \geq y\} = T_C, & \text{if } y = C, \\ \sup\{x \mid f(x) < y\} = +\infty, & \text{if } y > C, \end{cases}$$

and its upper pseudo-inverse $f_\uparrow^{-1}(y)$ is

$$f_\uparrow^{-1}(y) = \begin{cases} \inf\{x \mid f(x) > y\} = T_C, & \text{if } f(T_C) < y < C, \\ \sup\{x \mid f(x) \leq y\} = +\infty, & \text{if } y = C, \\ \sup\{x \mid f(x) \leq y\} = +\infty, & \text{if } y > C. \end{cases}$$

Otherwise, i.e., if $f(T_C) = C$*, its lower pseudo-inverse* $f_\downarrow^{-1}(y)$ *is*

$$f_\downarrow^{-1}(y) = \begin{cases} \inf\{x \mid f(x) \geq y\} \leq T_C, & \text{if } y < C, \\ \inf\{x \mid f(x) \geq y\} = T_C, & \text{if } y = C, \\ \sup\{x \mid f(x) < y\} = +\infty, & \text{if } y > C, \end{cases}$$

and its upper pseudo-inverse $f_\uparrow^{-1}(y)$ *is*

$$f_\uparrow^{-1}(y) = \begin{cases} \sup\{x \mid f(x) \leq y\} \leq T_C, & \text{if } y < C, \\ \sup\{x \mid f(x) \leq y\} = +\infty, & \text{if } y = C, \\ \sup\{x \mid f(x) \leq y\} = +\infty, & \text{if } y > C. \end{cases}$$

In other words, both pseudo-inverses are UI with $T_I = C$*.*

Proposition 14 *Let* $f \in \mathcal{U}$ *be a non-decreasing, UI function with* $T_I \in \mathbb{Q}_+$*. Then, its lower pseudo-inverse* $f_\downarrow^{-1}(y)$ *is*

$$f_\downarrow^{-1}(y) = \begin{cases} \inf\{x \mid f(x) \geq y\} < T_I, & \text{if } y < L, \\ \inf\{x \mid f(x) \geq y\} \leq T_I, & \text{if } y = L, \\ \inf\{x \mid f(x) \geq y\} = T_I, & \text{if } y > L, \end{cases}$$

and its upper pseudo-inverse $f_\uparrow^{-1}(y)$ *is*

$$f_\uparrow^{-1}(y) = \begin{cases} \sup\{x \mid f(x) \leq y\} < T_I, & \text{if } y < L, \\ \sup\{x \mid f(x) \leq y\} = T_I, & \text{if } y = L, \\ \sup\{x \mid f(x) \leq y\} = T_I, & \text{if } y > L. \end{cases}$$

In other words, both pseudo-inverses are UC with $T_C = L$*.*[7]

Starting from the above results, one can derive the few modifications to the algorithms described so far in this section to include these two corner cases. We leave this simple (yet tedious) task to the interested reader.

5 Composition of UPP Functions

In this section, we discuss the composition operator for UPP functions, i.e., $(f \circ g)(t) = f(g(t))$. Some explanations are due regarding the *physical meaning* of the above operation. In NC, functions are usually meant to map *time* to *bits*, hence one might legitimately wonder what the physical meaning of composition is in this setting. The answer largely depends on the object of a particular study. For instance, in the already quoted literature examples that employ composition (e.g., [23, 24]), g maps *bits* to *bits*. Specifically, f is the (strict) service curve of a link managed by a round-robin-like scheduler, and g carves out from f the (strict) service curve for *the flow under analysis*. As another

[7] The only exception being the (uninteresting) case of f such that $f(0) > 0$ and $T_I = 0$, for which $f_\downarrow^{-1}(y) = 0 \; \forall y \geq 0$.

example, [3, p. 128] shows that the horizontal deviation in Equation (2) can be computed as:

$$h(\alpha, \beta) = \sup_{t \geq 0} \left\{ \beta_{\downarrow}^{-1}(\alpha(t)) - t \right\}. \tag{15}$$

In the above, α maps time to bits, whereas $\beta_{\downarrow}^{-1}$ maps bits to time. Note that the above example also requires pseudo-inverses. We will show later on that pseudo-inversion is required in the algorithm for the composition operator.

This section is organized as follows. In Section 5.1 we show that the composition of UPP functions is again UPP, and provide expressions to compute its UPP parameters a priori. In Section 5.2 we discuss, first via an example and then via pseudocode, how to compute the composition algorithmically. In Section 5.3 we conclude with a summary of the by-curve algorithm and some observations on the algorithmic complexity of this operator.

5.1 Properties of composition of UPP functions

We assume that the inner function g is not UI.[8] We initially provide the result for generic f and g. Later on, we show that, if either or both are UA or UC, we can improve upon this result.

Theorem 15 *Let f and g be two functions $\in \mathcal{U}$ with g being non-negative, non-decreasing and not UI. Then, their composition $h \coloneqq f \circ g$ is again a function $\in \mathcal{U}$ with*

$$T_h = \max\left\{ g_{\downarrow}^{-1}(T_f), T_g \right\}, \tag{16}$$

$$d_h = p_{d_f} \cdot d_g \cdot q_{c_g}, \tag{17}$$

$$c_h = q_{d_f} \cdot p_{c_g} \cdot c_f, \tag{18}$$

where $p_{d_f}, p_{c_g} \in \mathbb{N}_0$, and $q_{d_f}, q_{c_g} \in \mathbb{N}$ such that $d_f = \frac{p_{d_f}}{q_{d_f}}$, and $c_g = \frac{p_{c_g}}{q_{c_g}}$. Note that $c_g \geq 0$ as g is non-decreasing.

Proof Let $k_h \in \mathbb{N}$ be arbitrary but fixed. Since g is UPP, it holds for all $t \geq T_g$ that

$$\begin{aligned} h(t + k_h \cdot d_h) &= f\left(g(t + k_h \cdot d_h)\right) \\ &= f\left(g\left(t + k_h \cdot \frac{d_h}{d_g} \cdot d_g\right)\right) \\ &\overset{(3)}{=} f\left(g(t) + k_h \cdot \frac{d_h}{d_g} \cdot c_g\right), \end{aligned}$$

where we used the UPP property of g in the last line. Note that $k_g \coloneqq k_h \cdot \frac{d_h}{d_g} \in \mathbb{N}$, since $\frac{d_h}{d_g} \overset{(17)}{=} p_{d_f} \cdot q_{c_g} \in \mathbb{N}$, where we used the fact that $d_f > 0$. Moreover, since f is UPP, too, we have under this additional assumption of $g(t) \geq T_f$ that

$$h(t + k_h \cdot d_h) = f\left(g(t) + k_h \cdot \frac{d_h}{d_g} \cdot c_g\right)$$

[8] If, for $t > T_I$, $g(t) = +\infty$ then $f(g(t)) = \lim_{y \to +\infty} f(y)$. The fact that f is UPP does not guarantee that such limit exists, e.g., when f is periodic.

$$
\begin{aligned}
&= f\left(g(t) + k_h \cdot \frac{d_h \cdot c_g}{d_g \cdot d_f} \cdot d_f\right) \\
&\overset{(3)}{=} f(g(t)) + k_h \cdot \frac{d_h \cdot c_g}{d_g \cdot d_f} \cdot c_f \\
&= h(t) + k_h \cdot \frac{d_h \cdot c_g \cdot c_f}{d_g \cdot d_f} \\
&\overset{(18)}{=} h(t) + k_h \cdot c_h.
\end{aligned}
$$

Note that $k_f := k_h \cdot \frac{d_h \cdot c_g}{d_g \cdot d_f} \in \mathbb{N}_0$, since $\frac{d_h \cdot c_g}{d_g \cdot d_f} \overset{(17)}{=} \frac{p_{d_f}}{d_f} \cdot q_{c_g} \cdot c_g = q_{d_f} \cdot p_{c_g} \in \mathbb{N}_0$ and we used that $c_g \geq 0$.

We set $t \geq T_g$ and $g(t) \geq T_f$, thus ensuring that both f and g are in their periodic part. Exploiting the notion of a lower pseudo-inverse and g being non-decreasing, the latter expression implies that $t \geq g_\downarrow^{-1}(T_f)$ [28, p. 62]. Therefore, we require

$$
t \geq T_h \overset{(16)}{=} \max\left\{g_\downarrow^{-1}(T_f), T_g\right\}.
$$

This concludes the proof. □

Remark 16 Note that the above is also true for the particular case in which $d_f \in \mathbb{N}, c_g \in \mathbb{N}_0$. In fact, it follows that $p_{d_f} = d_f$ and $q_{c_g} = 1$ and thus

$$
d_h \overset{(17)}{=} p_{d_f} \cdot q_{c_g} \cdot d_g = d_f \cdot d_g,
$$

and the properties are then verified since $\frac{d_h}{d_g} = d_f \in \mathbb{N}_0$ and $\frac{d_h \cdot c_g}{d_g \cdot d_f} = c_g \in \mathbb{N}_0$. The corresponding c_h is $c_f \cdot c_g$.

It follows from Theorem 15 that, in order to compute the representation R_h, we only need to compute $S_h^{D_h}$, where

$$
D_h = [0, T_h + d_h[= \left[0, \max\left\{g_\downarrow^{-1}(T_f), T_g\right\} + p_{d_f} \cdot d_g \cdot q_{c_g}\right[.
$$

It follows that

$$
S_h^{D_h} = S_f^{D_f} \circ S_g^{D_g},
$$

where

$$
\begin{aligned}
D_g &= [0, T_h + d_h[, \\
D_f &= \left[g(0), g\left((T_h + d_h)^-\right)\right].
\end{aligned} \tag{19}
$$

The reason D_f needs to be right-closed is that $S_g^{D_g}$ may end with a constant segment. If this happens, $\exists \bar{t} \in D_g$ such that $g(\bar{t}) = g\left((T_h + d_h)^-\right)$, thus we will need to compute $f\left(g\left(\bar{t}\right)\right) = f\left(g\left((T_h + d_h)^-\right)\right)$, and that is in fact the right boundary of D_f. On the other hand, if $S_g^{D_g}$ ends with a strictly increasing segment, it is safe to have D_f right-open.

Hereafter, we show that the above result can be improved when either or both functions are UA. First, we consider the case when only g is UA.

Proposition 17 *Let f and g be two functions $\in \mathcal{U}$ that are not UI, with g being non-negative, non-decreasing, UA, with $\rho_g > 0$ (hence not UC). Then, their composition $h := f \circ g$ is again a function $\in \mathcal{U}$ with*

$$
\begin{aligned}
T_h &= \max\left\{ g_{\downarrow}^{-1}(T_f), T_g \right\}, \\
d_h &= \frac{d_f}{\rho_g}, && (20) \\
c_h &= c_f. && (21)
\end{aligned}
$$

Proof Let $k_h \in \mathbb{N}$ be arbitrary but fixed. Since g is assumed to be UA, it holds for all $t \geq T_h$ that

$$
\begin{aligned}
h(t + k_h \cdot d_h) &= f\left(g(t + k_h \cdot d_h)\right) \\
&\overset{(4)}{=} f\left(g(T_h) + \rho_g \cdot (t + k_h \cdot d_h - T_h)\right) \\
&= f\left(\rho_g \cdot t + g(T_h) - \rho_g \cdot T_h + k_h \cdot d_f\right) \\
&\overset{(3)}{=} f\left(\rho_g \cdot t + g(T_h) - \rho_g \cdot T_h\right) + k_h \cdot c_f \\
&= f\left(g(T_h) + \rho_g \cdot (t - T_h)\right) + k_h \cdot c_f \\
&\overset{(4)}{=} f\left(g(t)\right) + k_h \cdot c_f \\
&= h(t) + k_h \cdot c_h.
\end{aligned}
$$

□

Again, in order to compute the representation R_h, we only need $S_h^{D_h}$, where

$$
D_h = [0, T_h + d_h[= \left[0, \max\left\{ g_{\downarrow}^{-1}(T_f), T_g \right\} + \frac{d_f}{\rho_g}\right[.
$$

It follows that

$$
S_h^{D_h} = S_f^{D_f} \circ S_g^{D_g},
$$

where

$$
\begin{aligned}
D_g &= [0, T_h + d_h[\\
&\overset{(20)}{=} \left[0, T_h + \frac{d_f}{\rho_g}\right[, \\
D_f &= \left[g(0), g\left((T_h + d_h)^-\right)\right[&& (22) \\
&\overset{(20)}{=} \left[g(0), g\left(T_h + \frac{d_f}{\rho_g}\right)\right[\\
&= [g(0), g(T_h) + d_f[.
\end{aligned}
$$

Here, we observe that domain D_f is smaller than the one obtained by applying directly Theorem 15, due to the disappearance of a factor $q_{d_f} \cdot p_{c_g} \geq 1$. In fact, with Theorem 15 we would have:

$$
D_g = [0, T_h + d_h[
$$

$$\begin{aligned}
&\overset{(17)}{=} \left[0, T_h + p_{d_f} \cdot d_g \cdot q_{c_g}\right[\\
&= \left[0, T_h + q_{d_f} \cdot p_{c_g} \cdot \frac{d_f}{\rho_g}\right[, \\
D_f &= \left[g(0), g\left((T_h + d_h)^-\right)\right[\\
&\overset{(17)}{=} \left[g(0), g\left(\left(T_h + p_{d_f} \cdot d_g \cdot q_{c_g}\right)^-\right)\right[\\
&= \left[g(0), g(T_h) + q_{d_f} \cdot p_{c_g} \cdot d_f\right[.
\end{aligned}$$

As specified in the statement of Proposition 17, we exclude the case when g is UC. This is because of Equation (20) where ρ_g is in the denominator, hence cannot be zero. However, if g is UC, a stronger proposition can be found as reported in Appendix D.

Next, we consider the case when only f is UA.

Proposition 18 *Let $f \in \mathcal{U}$ be UA and $g \in \mathcal{U}$ be non-negative, non-decreasing and not UI. Then, their composition $h := f \circ g$ is again $\in \mathcal{U}$ with*

$$T_h = \max\left\{g_\downarrow^{-1}(T_f), T_g\right\},$$

$$d_h = d_g, \tag{23}$$

$$c_h = c_g \cdot \rho_f. \tag{24}$$

Proof Let $k_h \in \mathbb{N}$ be arbitrary but fixed. Since f is assumed to be UA, it holds for all $t \geq T_h$ that

$$\begin{aligned}
h(t + k_h \cdot d_h) &= f\left(g(t + k_h \cdot d_h)\right) \\
&\overset{(3)}{=} f\left(g(t) + k_h \cdot c_g\right) \\
&\overset{(4)}{=} f(g(T_h)) + \rho_f \cdot (g(t) + k_h \cdot c_g - g(T_h)) \\
&= f(g(T_h)) + \rho_f \cdot (g(t) - g(T_h)) + k_h \cdot c_g \cdot \rho_f \\
&\overset{(4)}{=} f(g(t)) + k_h \cdot c_g \cdot \rho_f \\
&= h(t) + k_h \cdot c_h.
\end{aligned}$$

□

Again, for representation R_h, we only compute $S_h^{D_h}$, where

$$D_h = [0, T_h + d_h[= \left[0, \max\left\{g_\downarrow^{-1}(T_f), T_g\right\} + d_g\right[.$$

It follows that

$$S_h^{D_h} = S_f^{D_f} \circ S_g^{D_g},$$

where

$$\begin{aligned} D_g &= [0, T_h + d_h[\\ &\overset{(23)}{=} [0, T_h + d_g[, \\ D_f &= \left[g(0), g\left((T_h + d_h)^-\right)\right] \\ &\overset{(23)}{=} \left[g(0), g\left((T_h + d_g)^-\right)\right]. \end{aligned} \tag{25}$$

Again, domain D_g is smaller than the one that Theorem 15 would yield, due to the disappearance of a factor $p_{d_f} \cdot q_{c_g} \geq 1$. For comparison, Theorem 15 yields

$$\begin{aligned} D_g &= [0, T_h + d_h[\\ &\overset{(17)}{=} \left[0, T_h + p_{d_f} \cdot q_{c_g} \cdot d_g\right[, \\ D_f &= \left[g(0), g\left((T_h + d_h)^-\right)\right] \\ &\overset{(17)}{=} \left[g(0), g\left(\left(T_h + p_{d_f} \cdot d_g \cdot q_{c_g}\right)^-\right)\right]. \end{aligned}$$

When *both* functions are UA, we obtain a stronger result by showing that the composition is UA again.

Proposition 19 *Let $f \in \mathcal{U}$ and $g \in \mathcal{U}$ be UA functions with g being non-negative, non-decreasing and not UI. Then, their composition $h := f \circ g$ is again UA with*

$$\begin{aligned} T_h^a &= \max\left\{g_\downarrow^{-1}(T_f^a), T_g^a\right\}, \\ \rho_h &= \rho_f \cdot \rho_g. \end{aligned} \tag{26}$$

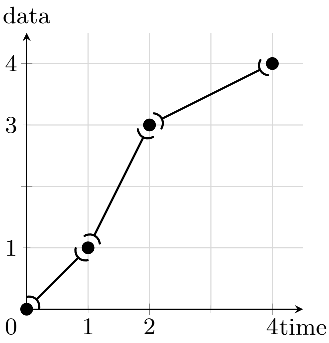
Proof If f is UI, the result is trivial. Let us assume that f is not UI. Define $T_h^a := \max\left\{g_\downarrow^{-1}(T_f^a), T_g^a\right\}$. Then we have that, for any $t \geq T_h^a$,

$$\begin{aligned} h(t + T_h^a) &= f\left(g(t + T_h^a)\right) \\ &\overset{(4)}{=} f\left(g(T_h^a) + \rho_g \cdot (t - T_h^a)\right) \\ &\overset{(4)}{=} f(g(T_h^a)) + \rho_f \cdot \left(\left(g(T_h^a) + \rho_g \cdot (t - T_h^a)\right) - g(T_h^a)\right) \\ &= f(g(T_h^a)) + \rho_f \cdot \rho_g \cdot (t - T_h^a) \\ &= h(T_h^a) + \rho_f \cdot \rho_g \cdot (t - T_h^a). \end{aligned}$$

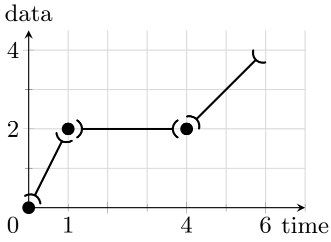
□

Considering Equation (19), we observe how taking these results into account will yield tighter D_f, D_g than what we obtain with Theorem 15.

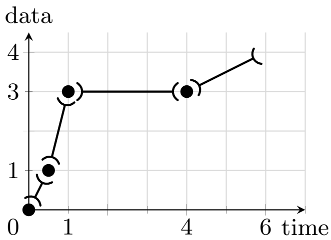
Finally, we mention that, if either or both f and g are UC, then the composition can be simplified further, even with respect to the above properties. We report the results in Appendix D.



(a) S_f



(b) S_g



(c) $S_f \circ S_g$

Fig. 7: Example of composition of two sequences.

5.2 By-sequence algorithm for composition

In this section, we discuss the by-sequence algorithm for the composition. Without loss of generality, we focus on sequences S_g, representing a non-negative and non-decreasing function g over an interval $[0, t[$, and S_f, representing a function f defined over the interval $[g(0), g(t^-)]$.[9] Then, $S_h = S_f \circ S_g$ is the sequence representing $h = f \circ g$ over the interval $[0, t[$. We use the example shown in Figure 7, where $t = 6$ and $g(t^-) = 4$.

First, we consider the shape of $f \circ g$ on an interval $]a, b[\subset [0, t[$, $a, b \in \mathbb{Q}_+$. Consider the case in which, for this interval, there exist $\rho_g, \rho_f \in \mathbb{Q}_+$ so that

$$\begin{aligned} g(x) &= g(a^+) + \rho_g \cdot (x - a), && \forall x \in \,]a, b[\,, \\ f(x) &= f\left(g(a^+)^+\right) + \rho_f \cdot \left(x - g(a^+)\right), && \forall x \in \left]g(a^+), g(b^-)\right[, \end{aligned} \tag{27}$$

where we use the shorthand notation

$$f\left(g(a^+)^+\right) = \lim_{x \to a^+} f\left(g(x)\right) = \lim_{y \to y_0} f(y),$$

[9] We consider D_f to be always right-closed since it yields the correct result for both cases discussed in the previous section. The right boundary of D_f is never used as a breakpoint in the algorithm anyway, as imposed by the condition $y_m < g(b^-)$ discussed below.

with $y_0 := \lim_{x \to a^+} g(x)$.

More broadly speaking, we have segment of g mapping to a segment of f. In the example of Figure 7, $]4, 6[$ is such an interval. Then, in this interval we can apply the chain rule and find that $h'(x) = f'(g(x)) \cdot g'(x) = \rho_g \cdot \rho_f$ for all $x \in]a, b[$. Thus, h is also a segment on $]a, b[$.

If either of the equations in (27) does not apply, it means that one function has one or more breakpoints over this interval. Assume initially that this is g. Let this finite sets of breakpoints be $t_0, \dots, t_n$, with $a < t_0 < \dots < t_n < b$. Then, the intervals $]a, t_0[, \dots,]t_n, b[$ verify the properties in Equation (27) while for any breakpoint t_i we can just compute $f(g(t_i))$. A similar reasoning can be done for f: consider the finite set of breakpoints $y_0, \dots, y_m$, with $g(a^+) < y_0 < \dots < y_m < g(b^-)$. Then, we can use the lower pseudo-inverse of g to find the corresponding $\bar{t}_i = g_\downarrow^{-1}(y_i)$.[10] The set $\{t_1, \dots, t_n\} \cup \{\bar{t}_1, \dots, \bar{t}_m\}$, preserving the ascending order, defines a finite set of breakpoints for $f \circ g$. Then, we have again a finite set of points $(t_i, f(g(t_i)))$, and open intervals for which we compute h as a segment with $\rho_h = \rho_f \cdot \rho_g$. In the example of Figure 7, $]0, 4[$ is such an interval:

- for S_g we find the set $\{t_1 = 1\}$;
- for S_f we find the set $\{y_1 = 1\} \to \{\bar{t}_1 = \frac{1}{2}\}$;
- the combined set of breakpoints is then $\left\{\frac{1}{2}, 1\right\}$, and the open intervals that verify Equation (27) is $\left\{]0, \frac{1}{2}[,]\frac{1}{2}, 1[,]1, 4[\right\}$.

By generalizing this reasoning, we obtain Algorithm 3.

Algorithm 3 Pseudocode for the composition of finite sequences

Input Two finite sequences of elements, S_f of f and S_g of g, so that S_g defined on $[0, a[$ and S_f defined on $[g(0), g(a^-)]$.

Return Composition $S_h = S_f \circ S_g$ consisting of a sequence of elements $O = \{o_1, \dots, o_m\}$.

Define an empty sequence of elements $O := \{\,\}$
Let $\mathbb{T}$ be an empty, but ordered set of distinct rationals
Let P_g be the set of points of S_g
for p_i **in** P_g **do**
    Add the time t_i of p_i to $\mathbb{T}$
end for
Let P_f be the set of points of S_f, excluding the last point $g(a^-)$
for p_i **in** P_f **do**
    Given time t_i of p_i, add $\bar{t}_i = g_\downarrow^{-1}(t_i)$ to $\mathbb{T}$ ▷ preserving the order in $\mathbb{T}$
end for
for each pair of consecutive (t_i, t_{i+1}) in $\mathbb{T}$ **do**
    Append $p := (t_i, f(g(t_i)))$ to O
    Append $s := \left(t_i, t_{i+1}, f\left(g(t_i^+)^+\right), f\left(g(t_{i+1}^-)^-\right)\right)$ to O
end for

[10] Following the discussion in Section 4, $(S_g)_\downarrow^{-1}$ is sufficient for this computation.

5.3 By-curve algorithm for composition

We can now discuss the by-curve algorithm, by combining the results presented in Sections 5.1 and 5.2. In Algorithm 4 we show the pseudocode to compute the composition $h = f \circ g$ of UPP functions f and g, in the most general case. The analogous for the more specialized cases, i.e., ultimately affine or ultimately constant operands, which here we omit for brevity, can be similarly derived by adjusting the parameter and domain computations.

Algorithm 4 Pseudocode for composition of UPP functions.

Input Representation R_f of a UPP function f, consisting of sequence S_f and parameters T_f, d_f and c_f; Representation R_g of a non-negative and non-decreasing UPP function g, consisting of sequence S_g and parameters T_g, d_g and c_g.

Return Representation R_h of $h = f \circ g$.

Compute the UPP parameters for the result ▷ Theorem 15
    $T_h \leftarrow \max\left\{ g_{\downarrow}^{-1}(T_f), T_g \right\}$
    $d_h \leftarrow p_{d_f} \cdot d_g \cdot q_{c_g}$
    $c_h \leftarrow q_{d_f} \cdot p_{c_g} \cdot c_f$
Compute $S_f^{D_f}$ and $S_g^{D_g}$ ▷ Equation (19)
    $D_f \leftarrow [g(0), g\left((T_h + d_h)^-\right)[$
    $S_f^{D_f} \leftarrow \mathrm{Cut}(R_f, D_f)$
    $D_g \leftarrow [0, T_h + d_h[$
    $S_g^{D_g} \leftarrow \mathrm{Cut}(R_g, D_g)$
Compute $S_h \leftarrow S_f^{D_f} \circ S_g^{D_g}$ ▷ Algorithm 3
$R_h \leftarrow (S_h, T_h, d_h, c_h)$

Regarding the complexity of Algorithm 4, we note that the main cost is computing $S_h \leftarrow S_f^{D_f} \circ S_g^{D_g}$. Since Algorithm 3 is a linear scan of $S_f^{D_f}$ and $S_g^{D_g}$, the resulting complexity is $\mathcal{O}\left(n\left(S_f^{D_f}\right) + n\left(S_g^{D_g}\right)\right)$. Note that given the expressions in Theorem 15, this computational cost highly depends on the numerical properties of the operands, i.e., numerators and denominators of UPP parameters, rather than simply the cardinalities of R_f and R_g. Thus, using the specialized properties of Propositions 17 to 19 yields performance improvements, since D_f and D_g are smaller.

We remark again that the result of the composition may have a non-minimal representation (see the discussion at the end of Section 4.1).

6 Proof of Concept

In this section, we show how the algorithms presented in this paper allow one to replicate the result appeared in a recent NC paper [23].

Table 2: Performance comparison of composition with and without UA optimization.

Runtime	Not optimized	Optimized
75th percentile	1117.72 ms	0.67 ms
median	1105.01 ms	0.55 ms
25th percentile	1088.61 ms	0.50 ms

The algorithms described in this paper, including variants and corner cases omitted for brevity, are implemented in the publicly available Nancy NC library [30]. Nancy is a C# library implementing the UPP model and its operators, as described in [19, 20]. Moreover, it implements state-of-the-art algorithms that improve the efficiency of NC operators, described in [31], and lower pseudo-inverse, upper pseudo-inverse and composition operators, described in this paper. Nancy makes extensive use of parallelism. However, the NC operators described in this paper are implemented as sequential.

As a notable example, we implemented the results from [23, Theorem 1], which uses the composition operator, using the same parameters of the example in [23, Figure 3]. The above theorem allows us to compute the service curve for a flow served by an Interleaved Weighted Round-Robin scheduler, once a) the weight of the flow; b) the minimum and maximum packet length for each flow, and c) the (strict) service curve for the entire aggregate of flows $\beta(t)$ are known. The complete formulation of the theorem – which is rather cumbersome – is postponed to Appendix E. For the purpose of this proof of concept, the important bit is that computing the service curve of the flow involves computing a function γ_i that takes into account flow i's characteristics (e.g., weight, packet lengths), and then, given β as the (strict) service curve of the server regulated by IWRR, computing the (strict) per-flow service curve for flow i as $\beta_i = \gamma_i \circ \beta$.[11] In the example in [23, Figure 3], β is a constant-rate service curve, thus UA, while γ_i is, in general, a UPP function. On the one hand, this confirms that limiting NC algorithms to UA curves only is severely constraining – in this example, one could not compute flow i's service curve without an algorithm that handles UPP curves. On the other hand, it means that we can obtain the same result by applying both Theorem 15 and its specialized version for UA inner functions Proposition 17, and that we can expect the latter to be more efficient due to the tighter D_f, as explained below Equation (22).

Our experiments confirm the above intuition. We run the computation on a laptop computer (i7-10750H, 32 GB RAM). As shown in Table 2, when using Theorem 15, computing the result took a median of 1.11 seconds. On the other hand, using Proposition 17 the same result is obtained in 0.55 milliseconds in the median, an improvement of three orders of magnitude. Listing 1 and Figure 8 report, respectively, the code used and the resulting plot.

[11] Recall that composition requires the lower pseudo-inverse of the inner function to be computed, hence this example makes use of both the algorithms presented in this paper.

Listing 1 Code used to replicate the results of [23, Theorem 1].

```
var weights = new []{4, 6, 7, 10};
var l_min = new []{4096, 3072, 4608, 3072};
var l_max = new []{8704, 5632, 6656, 8192};
var beta = new RateLatencyServiceCurve(
    rate: 10000, // 10 Mb/s, but using ms as time unit
    latency: 0
);
var unit_rate = new RateLatencyServiceCurve(1, 0);

int Phi_i_j(int i, int j, int x) {...}

int Psi_i(int i, int x) {...}

int L_tot(int i) {...}

int i = 0; // the flow of interest
var stairs = new List<Curve>();
for(int k = 0; k < weights[i]; k++)
{
    var stair = new StairCurve(l_min[i], L_tot(i));
    var delayed_stair = stair.DelayBy(Psi_i(i, k * l_min[i]));
    stairs.Add( delayed_stair );
}
var U_i = Curve.Addition(stairs); // summation of min-plus curves
var gamma_i = Curve.Convolution(unit_rate, U_i);
var beta_i = Curve.Composition(gamma_i, beta);
```

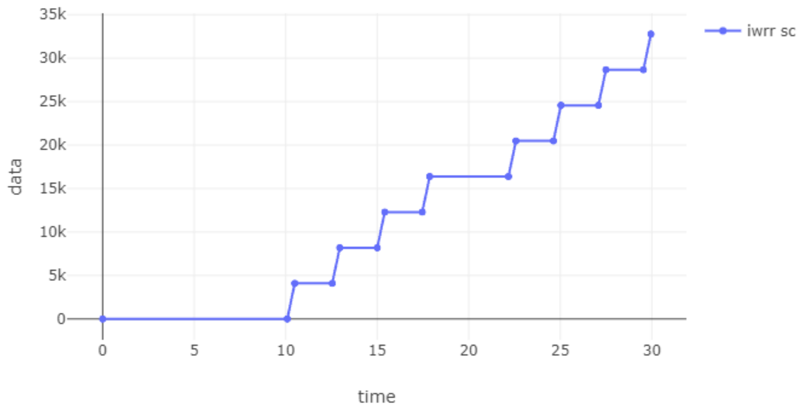


Fig. 8: Plot of the resulting service curve β_i.

It is worth noting that [24, Theorem 1] describes a similar result for the Deficit Round-Robin scheduler, under similar hypotheses, still making use of composition, with the outer curve being non-UA. The derivations in this section apply to this case as well, with minimal obvious modifications. Several other results in [24] make use of composition as well.

7 Conclusions

Automated computation of Network Calculus operations is necessary to carry out analyses of non-trivial network scenarios. Therefore, algorithms that transform representations of operand functions into result functions are required for each "useful" NC operator. Recently, pseudo-inverses and composition operators have been repeatedly used in NC papers. To the best of our knowledge, these operators lacked an algorithmic description that would allow their implementation in software.

This paper fills the above gap, by providing algorithms for lower and upper pseudo-inverses and composition of operators. We have presented algorithms that work under general assumptions (i.e., UPP operands), as well as specialized ones that leverage the fact that operands are UA (or UC) to compute results faster. We have discussed the complexity of these algorithms, as well as corner cases, with a rigorous mathematical exposition. Beside the theoretical contribution, we provided a practical one by including the above algorithms (together with the others known from the literature) in an open-source free library called Nancy. This allows researchers to experiment with our results to study complex scenarios, or support the generation of novel theoretical insight.

Future work on this topic will include studying the computational and numerical properties of NC operators. As the example in Section 6 shows (not to mention those in [31], by the same authors), exploiting more knowledge on the operands allows one to compute the same results via specialized versions of the algorithms, often in considerably shorter times (some by orders of magnitude). We believe that this is an avenue of research worth pursuing, with the aim of enabling larger-scale real-world performance studies.

Acknowledgments. This work was partially supported by the Italian Ministry of Education and Research (MIUR) in the framework of the CrossLab project (Departments of Excellence), and by the University of Pisa, through grant "Analisi di reti complesse: dalla teoria alle applicazioni" - PRA 2020.

Compliance with Ethical Standards

The authors declare that they have no conflict of interest.

References

[1] Cruz, R.L.: A calculus for network delay, part I: Network elements in isolation. IEEE Transactions on information theory **37**(1), 114–131 (1991)

[2] Cruz, R.L.: A calculus for network delay, part II: Network analysis. IEEE Transactions on information theory **37**(1), 132–141 (1991)

[3] Le Boudec, J.-Y., Thiran, P.: Network Calculus: a Theory of Deterministic Queuing Systems for the Internet. Springer, Berlin, Germany (2001)

[4] Chang, C.-S.: Performance Guarantees in Communication Networks. Springer, New York, USA (2000)

[5] Le Boudec, J.-Y.: Application of network calculus to guaranteed service networks. IEEE Transactions on Information theory **44**(3), 1087–1096 (1998)

[6] Firoiu, V., Le Boudec, J.-Y., Towsley, D., Zhang, Z.-L.: Theories and models for internet quality of service. Proceedings of the IEEE **90**(9), 1565–1591 (2002)

[7] Le Boudec, J.-Y.: Application of network calculus to guaranteed service networks. IEEE Transactions on Information Theory **44**(3), 1087–1096 (1998). https://doi.org/10.1109/18.669170

[8] Bennett, J.C., Benson, K., Charny, A., Courtney, W.F., Le Boudec, J.-Y.: Delay jitter bounds and packet scale rate guarantee for expedited forwarding. IEEE/ACM Transactions on networking **10**(4), 529–540 (2002)

[9] Fidler, M., Sander, V.: A parameter based admission control for differentiated services networks. Computer Networks **44**(4), 463–479 (2004)

[10] Schmitt, J.B., Roedig, U.: Sensor network calculus–a framework for worst case analysis. In: International Conference on Distributed Computing in Sensor Systems, pp. 141–154 (2005). Springer

[11] Charara, H., Scharbarg, J.-L., Ermont, J., Fraboul, C.: Methods for bounding end-to-end delays on an AFDX network. In: 18th Euromicro Conference on Real-Time Systems (ECRTS'06), p. 10 (2006). IEEE

[12] Bauer, H., Scharbarg, J.-L., Fraboul, C.: Worst-case end-to-end delay analysis of an avionics AFDX network. In: 2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010), pp. 1220–1224 (2010). IEEE

[13] Zhang, J., Chen, L., Wang, T., Wang, X.: Analysis of TSN for industrial automation based on network calculus. In: 2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), pp. 240–247 (2019). IEEE

[14] Maile, L., Hielscher, K.-S., German, R.: Network Calculus results for TSN: An introduction. In: 2020 Information Communication Technologies Conference (ICTC), pp. 131–140 (2020). IEEE

[15] Zhao, L., Pop, P., Zheng, Z., Daigmorte, H., Boyer, M.: Latency analysis of multiple classes of AVB traffic in TSN with standard credit behavior

using Network Calculus. IEEE Trans. Ind. Electron. **68**(10), 10291–10302 (2021). https://doi.org/10.1109/TIE.2020.3021638

[16] Rehm, F., Seitter, J., Larsson, J.-P., Saidi, S., Stea, G., Zippo, R., Ziegenbein, D., Andreozzi, M., Hamann, A.: The road towards predictable automotive high-performance platforms. In: 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 1915–1924 (2021). IEEE

[17] Andreozzi, M., Conboy, F., Stea, G., Zippo, R.: Heterogeneous systems modelling with adaptive traffic profiles and its application to worst-case analysis of a DRAM controller. In: 2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC), pp. 79–86 (2020). IEEE

[18] Boyer, M., Graillat, A., de Dinechin, B.D., Migge, J.: Bounding the delays of the MPPA network-on-chip with network calculus:models and benchmarks. Perform. Evaluation **143**, 102–124 (2020). https://doi.org/10.1016/j.peva.2020.102124

[19] Bouillard, A., Thierry, É.: An algorithmic toolbox for network calculus. Discrete Event Dynamic Systems **18**(1), 3–49 (2008)

[20] Bouillard, A., Boyer, M., Le Corronc, E.: Deterministic Network Calculus: From Theory to Practical Implementation. Wiley, Hoboken, NJ (2018)

[21] Bouillard, A., Cottenceau, B., Gaujal, B., Hardouin, L., Lagrange, S., Lhommeau, M., Thierry, E.: COINC library: a toolbox for the network calculus

[22] RealTime-at-Work: RTaW-Pegase (min,+) library. https://www.realtimeatwork.com/rtaw-pegase-libraries/. Accessed: 2022-04-05

[23] Tabatabaee, S.M., Le Boudec, J.-Y., Boyer, M.: Interleaved Weighted Round-Robin: A Network Calculus Analysis. IEICE Transactions on Communications **104**(12), 1479–1493 (2021)

[24] Tabatabaee, S.M., Le Boudec, J.-Y.: Deficit round-robin: A second network calculus analysis. IEEE/ACM Transactions on Networking (2022)

[25] Mohammadpour, E., Stai, E., Boudec, J.-Y.L.: Improved delay bound for a service curve element with known transmission rate. IEEE Networking Letters **1**(4), 156–159 (2019)

[26] Mohammadpour, E., Stai, E., Boudec, J.-Y.L.: Improved Network Calculus delay bounds in Time-Sensitive Networks. arXiv preprint arXiv:2204.10906 (2022)

[27] IEEE: Time-sensitive networking (TSN) task group. [Online]. https://1.ieee802.org/tsn/. Accessed: 2022-05-16

[28] Liebeherr, J.: Duality of the max-plus and min-plus network calculus. Foundations and Trends in Networking **11**(3-4), 139–282 (2017). https://doi.org/10.1561/1300000059

[29] Pollex, V., Lipskoch, H., Slomka, F., Kollmann, S.: Runtime improved computation of path latencies with the real-time calculus. In: Proceedings of the 1st International Workshop on Worst-Case Traversal Time, pp. 58–65 (2011)

[30] Zippo, R., Stea, G.: Nancy: an efficient parallel Network Calculus library. http://arxiv.org/abs/2205.11449 (2022)

[31] Zippo, R., Stea, G.: Computationally efficient worst-case analysis of flow-controlled networks with Network Calculus. https://arxiv.org/abs/2203.02497 (2022)

[32] Boyer, M., Stea, G., Sofack, W.M.: Deficit Round Robin with network calculus. In: 6th International ICST Conference on Performance Evaluation Methodologies and Tools, Cargese, Corsica, France, October 9-12, 2012, pp. 138–147 (2012). https://doi.org/10.4108/valuetools.2012.250202. https://doi.org/10.4108/valuetools.2012.250202

[33] Bondorf, S., Schmitt, J.B.: The DiscoDNC v2 – A Comprehensive Tool for Deterministic Network Calculus. In: Proc. of the International Conference on Performance Evaluation Methodologies and Tools. ValueTools '14, pp. 44–49 (2014). https://dl.acm.org/citation.cfm?id=2747659

[34] Guan, N., Yi, W.: Finitary real-time calculus: Efficient performance analysis of distributed embedded systems. In: 2013 IEEE 34th Real-Time Systems Symposium, pp. 330–339 (2013)

[35] Lampka, K., Bondorf, S., Schmitt, J.B., Guan, N., Yi, W.: Generalized Finitary Real-Time Calculus. In: Proc. of the 36th IEEE International Conference on Computer Communications (INFOCOM 2017) (2017)

A Properties of Ultimately Affine (UA) Functions

Proposition 20 *A function $f \in \mathcal{U}$ is Ultimately Affine (UA) (defined in Equation (4)) iff there exist $T \in \mathbb{Q}_+, \sigma, \rho \in \mathbb{Q}$ such that either*

$$f(t) = \rho t + \sigma \quad \forall t \geq T \tag{28}$$

or if $f(t) = -\infty$ or $f(t) = +\infty$ for all $t \geq T$.

Proof The proof is trivial for f being $-\infty$ or $+\infty$ for all $t \geq T$. Therefore, we limit ourselves to the cases of f being finite.

"$\Rightarrow$"

Let f be UA. Define $T := T_f^a$, $\sigma := f\left(T_f^a\right) - \rho_f \cdot T_f^a$ and $\rho := \rho_f$. Then, it holds for all $t \geq T$ that

$$f(t) \overset{(4)}{=} \left(f\left(T_f^a\right) - \rho_f \cdot T_f^a\right) + \rho_f \cdot t = \sigma + \rho \cdot t.$$

"$\Leftarrow$" Assume f to verify the condition in Equation (28). Therefore, assume that $f(t) = \rho t + \sigma$ for all $t \geq T$. Define $T_f^a := T$, $\rho_f := \rho$. Then for all $t \geq T_f^a$

$$\begin{aligned} f(t) &= f\left((t - T_f^a) + T_f^a\right) \\ &\overset{(28)}{=} \sigma + \rho_f \left((t - T_f^a) + T_f^a\right) \\ &= \left(\rho_f T_f^a + \sigma\right) + \rho_f \cdot (t - T_f^a) \\ &\overset{(28)}{=} f(T_f^a) + \rho_f (t - T_f^a). \end{aligned}$$

This concludes the proof. □

B Differences in Pseudo-Inverses Definitions

In [28, p. 60], which considers functions from $\mathbb{R} \to \mathbb{R}$, lower and upper pseudo-inverses are introduced as

$$\begin{aligned} f_\downarrow^{-1}(y) &= \inf\left\{t \mid f(t) \geq y\right\} = \sup\left\{t \mid f(t) < y\right\}, \\ f_\uparrow^{-1}(y) &= \sup\left\{t \mid f(t) \leq y\right\} = \inf\left\{t \mid f(t) > y\right\}. \end{aligned}$$

However, when one considers a domain bounded from below by 0, such as in our case, the rightmost equalities do not hold for $y \leq f(0)$. As a counterexample, consider $y = f(0)$. Then,

$$\begin{aligned} f_\downarrow^{-1}(y) &= \inf\left\{x \geq 0 \mid f(x) \geq y\right\} = 0, \\ f_\downarrow^{-1}(y) &= \sup\left\{x \geq 0 \mid f(x) < y\right\} = \sup\left\{\emptyset\right\} = -\infty. \end{aligned}$$

Proposition 8 states a weaker form of equivalence for functions in $\mathcal{U}$. We provide here a proof.

Proof The proof follows mostly along the lines of Lemma 3.2 in [20, pp. 46].[12]

1. Lower pseudo-inverse: first, note that $\{t \geq 0 \mid f(t) < y\}$ and $\{t \geq 0 \mid f(t) \geq y\}$ form a partition of $\mathbb{Q}_+$. Moreover, as f is non-decreasing and $y > f(0)$, $\{t \geq 0 \mid f(t) \geq y\}$ is a non-empty interval of the form $[b, +\infty[$ or $]b, +\infty[$ for some $b > 0$. As a consequence, $\{t \geq 0 \mid f(t) < y\}$ is a non-empty interval of the form $[0, b[$ or $[0, b]$. Thus, we have $b = \inf\{t \geq 0 \mid f(t) \geq y\} = \sup\{t \geq 0 \mid f(t) < y\}$.
2. Upper pseudo-inverse: for $y > f(0)$, the proof is the almost same as in 1., we just replace $\{t \geq 0 \mid f(t) < y\}$ by $\{t \geq 0 \mid f(t) \leq y\}$ and $\{t \geq 0 \mid f(t) \geq y\}$ by $\{t \geq 0 \mid f(t) > y\}$. Then, $b = \sup\{t \geq 0 \mid f(t) \leq y\} = \inf\{t \geq 0 \mid f(t) > y\}$.

   Next, consider the case $y = f(0)$. Let us define $t_1 \coloneqq \sup\{t \geq 0 \mid f(t) = f(0)\} \in \mathbb{Q}_+ \cup \{+\infty\}$. It holds that

$$\begin{aligned} \sup\{t \geq 0 \mid f(t) \leq y\} &= \sup\{t \geq 0 \mid f(t) \leq f(0)\} \\ &= t_1 \end{aligned}$$

   as well as

$$\begin{aligned} \inf\{t \geq 0 \mid f(t) > y\} &= \inf\{t \geq 0 \mid f(t) > f(0)\} \\ &= \inf\{t \geq t_1 \mid f(t) > f(0)\} \\ &= t_1, \end{aligned}$$

   where we used in the second line that t_1 is a lower bound for the set $\{t \geq 0 \mid f(t) > f(0)\}$.

□

C Calculation of Lower and Upper Pseudo-Inverses

We report here the rigorous mathematical derivations for cases c1-c8 in Table 1.

Point after segment (cases c1-c4)

In these cases we have, in general, an f such that

$$f(x) = \begin{cases} b_1 + \rho\,(x - t_1), & \text{if } t_1 < x < t_2, \\ b_2, & \text{if } x \to t_2^-, \\ b_3, & \text{if } x = t_2. \end{cases}$$

Since f is non-decreasing, $b_1 + \rho\,(x - t_1) \leq b_2 \leq b_3$ for all $x \in \,]t_1, t_2[$.

We then distinguish four cases based on two properties:

- Whether or not the segment is constant, i.e., $\rho = 0 \rightarrow b_1 = b_2$;
- Whether or not there is a discontinuity at t_2, i.e., $b_2 < b_3$.

[12] We note however that Lemma 3.2 in [20, pp. 46] is incomplete, since it does not account for the case $y \leq f(0)$ – see our counterexample above.

Case c1: $\rho = 0$ ***and*** $b_1 = b_2 < b_3$ ***(constant segment followed by a discontinuity).***

It holds that

$$f_{\downarrow}^{-1}(y) = \begin{cases} \inf \{x \mid f(x) \geq y\} = t_2, & \text{if } b_1 < y < f(t_2) = b_2, \\ \inf \left\{ x \mid f(x) \geq \underbrace{y}_{=b_2} \right\} = t_2, & \text{if } y = f(t_2) = b_2, \end{cases} \tag{29}$$

and

$$f_{\uparrow}^{-1}(y) = \begin{cases} \sup \left\{ x \mid f(x) \leq \overbrace{y}^{=b_1} \right\} = t_2, & \text{if } y = b_1 = f(t_1^+), \\ \sup \{x \mid f(x) \leq y\} = t_2, & \text{if } b_1 = f(t_1^+) < y < f(t_2), \\ \inf \left\{ x \mid \underbrace{f(x)}_{=b_2} > y \right\} = \sup \{x \mid f(x) \leq y\} = t_2, & \text{if } y = f(t_2) = b_2. \end{cases} \tag{30}$$

Case c2: $\rho = 0$ ***and*** $b_1 = b_2 = b_3$ ***(constant segment without any discontinuity).***

It holds that

$$f_{\downarrow}^{-1}(y) = \inf \left\{ x \mid \overbrace{f(x)}^{=b_1} \geq y \right\} = t_1, \text{if } y = b_1. \tag{31}$$

However, we do not add a value as it is processed in the "segment after point" section. Moreover,

$$f_{\uparrow}^{-1}(y) := \sup \left\{ x \mid \underbrace{f(x)}_{=b_1} \leq y \right\} = t_2, \text{if } y = b_1. \tag{32}$$

Case c3: $\rho > 0$ ***and*** $b_2 < b_3$ ***(non-constant segment followed by a discontinuity).***

$$f_{\downarrow}^{-1}(y) = \begin{cases} \inf \{x \mid f(x) \geq y\} = t_2, & \text{if } y = b_1 + r\,(t_2 - t_1) = f(t_2^-), \\ \inf \{x \mid f(x) \geq y\} = t_2, & \text{if } f(t_2^-) < y < f(t_2) = b_3, \\ \inf \left\{ x \mid f(x) \geq \underbrace{y}_{=b_3} \right\} = t_2, & y = f(t_2) = b_3, \end{cases} \tag{33}$$

and

$$f_{\uparrow}^{-1}(y) = \begin{cases} \inf\{x \mid f(x) > y\} = t_2, & \text{if } y = b_1 + r(t_2 - t_1) = f(t_2^-), \\ \inf\{x \mid f(x) > y\} = t_2, & \text{if } f(t_2^-) < y < f(t_2) = b_3, \\ \inf\left\{x \mid f(x) > \underbrace{y}_{=b_3}\right\} = t_2, & y = f(t_2) = b_3. \end{cases} \tag{34}$$

Case c4: $\rho > 0$ ***and*** $b_2 = b_3$ ***(non-constant segment without any discontinuity).***

$$f_{\downarrow}^{-1}(y) = \inf\left\{x \mid f(x) \geq \underbrace{y}_{=b_2}\right\} = t_2, \; y = f(t_2) = b_2, \tag{35}$$

and

$$f_{\uparrow}^{-1}(y) = \inf\left\{x \mid f(x) > \underbrace{y}_{=b_2}\right\} = t_2, \; y = f(t_2) = b_2. \tag{36}$$

Segment after point (cases 5-8)

In these cases we have, in general, an f such that

$$f(x) = \begin{cases} b_1, & x = t_1, \\ b_2, & x \to t_1^+, \\ b_2 + \rho(x - t_1), & t_1 < x < t_2. \end{cases}$$

We then distinguish four cases based on two properties:

- Whether or not the segment is constant, i.e., $\rho = 0 \to b_2 = b_3$;
- Whether or not there is a discontinuity at t_1, i.e., $b_1 \neq b_2$.

Case c5: $b_1 < b_2 = b_3$ ***and*** $\rho = 0$ ***(discontinuity followed by a constant segment).***

It holds that

$$f_{\downarrow}^{-1}(y) = \begin{cases} \inf\{x \mid f(x) \geq y\} = \sup\{x \mid f(x) < y\} = t_1, & \text{if } b_1 < y < f(t_1^+) = b_2, \\ \inf\left\{x \mid f(x) \geq \underbrace{y}_{=b_2}\right\} = t_1, & \text{if } y = f(t_1^+) = b_2, \end{cases} \tag{37}$$

and

$$f_{\uparrow}^{-1}(y) = \begin{cases} \inf\left\{x \mid f(x) > y\right\} = \sup\left\{x \mid f(x) \leq y\right\} = t_1, & \text{if } b_1 < y < f(t_1^+) = b_2, \\ \inf\left\{x \mid f(x) > y\right\} = \sup\left\{x \mid f(x) \leq \underbrace{y}_{=b_2}\right\} = t_2, & \text{if } y = f(t_1^+) = b_2. \end{cases} \tag{38}$$

Case c6: $b_1 = b_2 = b_3$ ***and*** $\rho = 0$ ***(no discontinuity and a constant segment).***
Then it holds that

$$f_{\downarrow}^{-1}(y) = \inf\left\{x \mid \overbrace{f(x)}^{=b_1} \geq y\right\} = t_1, \text{if } y = b_1. \tag{39}$$

However, we do not add a value as it is processed in the "point after segment" section. Moreover,

$$f_{\uparrow}^{-1}(y) := \sup\left\{x \mid \underbrace{f(x)}_{=b_1} \leq y\right\} = t_2, \text{if } y = b_1. \tag{40}$$

Case c7: $b_1 < b_2$ ***and*** $\rho > 0$ ***(discontinuity followed by a non-constant segment).***
We have

$$f_{\downarrow}^{-1}(y) = \begin{cases} \inf\left\{x \mid f(x) \geq y\right\} = t_1, & \text{if } b_1 < y < f(t_1^-) = b_2, \\ \inf\left\{x \mid f(x) \geq \underbrace{y}_{=b_2}\right\} = t_1, & \text{if } y = f(t_1^-) = b_2, \\ \inf\left\{x \mid b_2 + \rho\,(x - t_1) \geq y\right\} = t_1 + \frac{y - b_2}{\rho}, & \text{if } b_2 < y < b_3, \end{cases} \tag{41}$$

and

$$f_{\uparrow}^{-1}(y) = \begin{cases} \inf\left\{x \mid f(x) > y\right\} = t_1, & \text{if } b_1 < y < b_2, \\ \inf\left\{x \mid f(x) > y\right\} = \sup\left\{x \mid f(x) \leq \underbrace{y}_{=b_2}\right\} = t_1, & \text{if } y = b_2, \\ \inf\left\{x \mid b_2 + \rho\,(x - t_1) > y\right\} = t_1 + \frac{y - b_2}{\rho}, & \text{if } b_2 < y < b_3. \end{cases} \tag{42}$$

Case c8: $b_1 = b_2$ ***and*** $\rho > 0$ ***(no discontinuity and non-constant segment).***

We have

$$f_{\downarrow}^{-1}(y) = \inf\left\{x \mid b_1 + \rho\,(x - t_1) \geq y\right\} = t_1 + \frac{y - b_1}{\rho}, \quad \text{if } b_1 = f(t_1^+) < y < f(t_2^-) = b_2, \tag{43}$$

and

$$f_{\uparrow}^{-1}(y) = \inf\left\{x \mid b_1 + \rho\,(x - t_1) > y\right\} = t_1 + \frac{y - b_1}{\rho}, \quad \text{if } b_1 = f(t_1^+) < y < f(t_2^-) = b_2. \tag{44}$$

D Composition of Ultimately Constant (UC) Functions

Proposition 21 *Let f and g be two functions $\in \mathcal{U}$ that are not UI, with g being non-negative, non-decreasing and UC. Then, their composition $h := f \circ g$ is again UC with*

$$T_h = T_g. \tag{45}$$

Proof For $t \geq T_g$, it holds that

$$h(t) = f(g(t)) = f(g(T_g)) = h(T_h).$$

□

Proposition 22 *Let f be UC and g be a function $\in \mathcal{U}$ that is non-negative and non-decreasing. Then, their composition $h := f \circ g$ is again UC with*

$$T_h = g_{\downarrow}^{-1}(T_f). \tag{46}$$

Proof For $t \geq g_{\downarrow}^{-1}(T_f)$, it holds that

$$h(t) = f(g(t)) = f(T_f) = h(T_h).$$

□

Proposition 23 *Let f and g be UC functions, with g being non-negative and non-decreasing. Then, their composition $h := f \circ g$ is again UC with*

$$T_h = \min\left\{T_f, g_{\downarrow}^{-1}(T_f)\right\}. \tag{47}$$

Proof The proof is simply a combination of the previous two propositions. □

E Service Curve of a Flow in Interleaved Weighted Round Robin

We report here the statement of Theorem 1 in [23], for ease of reference. We slightly rephrased it to aid comprehension.

Theorem 24 (Strict Per-Flow Service Curves for IWRR) *Assume n flows arriving at a server performing interleaved weighted round robin (IWRR) with weights $w_1, \ldots, w_n$. Let $l_i^{\min}$ and $l_j^{\max}$ denote the minimum and maximum packet size of the respective flow. Let this server offer a super-additive strict service curve β to these n flows. Then,*

$$\beta^i(t) := \gamma_i\left(\beta(t)\right)$$

is a strict service curve for flow f_i, where

$$\begin{aligned}
\gamma_i(t) &:= \beta_{1,0} \otimes U_i\left(t\right), \\
U_i(t) &:= \sum_{k=0}^{w_i - 1} \nu_{l_i^{\min}, L_{\text{tot}}} \left(\left[t - \psi_i\left(k l_i^{\min}\right) \right]^+ \right), \\
L_{\text{tot}} &:= w_i l_i^{\min} + \sum_{j : j \neq i} w_j l_j^{\max}, \\
\psi_i(x) &:= x + \sum_{j \neq i} \phi_{ij} \left(\left\lfloor \frac{x}{l_i^{\min}} \right\rfloor \right) l_j^{\max}, \\
\phi_{ij}(p) &:= \left\lfloor \frac{p}{w_i} \right\rfloor w_j + \left[w_j - w_i \right]^+ + \min\left\{ (p \mod w_i) + 1, w_j \right\},
\end{aligned}$$

$\beta_{1,0}$ is a constant-rate function with slope 1, and the stair function $\nu_{h,P}(t)$ is defined as

$$\nu_{h,P}(t) := h \left\lceil \frac{t}{P} \right\rceil, \qquad \text{for } t \geq 0.$$