UNIVERSITÀ DEGLI STUDI FIRENZE

PhD Program in Smart Computing
Dipartimento di Ingegneria dell'Informazione (DINFO)

Analysis of Algorithmic and Computational Aspects of Deterministic Network Calculus

Raffaele Zippo

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Smart Computing

PhD Program in Smart Computing
University of Florence, University of Pisa, University of Siena

Analysis of Algorithmic and Computational Aspects of Deterministic Network Calculus

Raffaele Zippo

Advisor:

Prof. Giovanni Stea

Head of the PhD Program:

Prof. Stefano Berretti

Evaluation Committee:
Dr. Anne Bouillard, *Huawei*
Prof. Marc Boyer, *ONERA*

XXXV ciclo — January 2023

A Lucrezia

Acknowledgments

This work would have not been possible without the help, support and advice of many, whom I am grateful to.

I want to thank my family for supporting and encouraging me throughout my life. I am grateful to my girlfriend, Lucrezia, for being close when I needed it the most. I am grateful to all my friends for the fun times, the unfun times, and for bearing with me through both.

I am thankful to my advisor, Prof. Giovanni Stea, for being a great mentor and example. Your continuous attention and dedication are admirable.

I would like to thank all my coauthors for their collaboration and for sharing their knowledge and advice, namely Dr. Matteo Andreozzi, Prof. Antonio Frangioni, Prof. Laura Galli, Dr. Paul Nikolaus, Prof. Giovanni Stea.

I want to thank my PhD committee, namely Dr. Anne Bouillard and Prof. Marc Boyer, for their time and consideration to review this thesis, and their constructive comments that helped me improve it considerably.
I want to thank the members of my PhD award panel, namely Prof. Enrico Bini, Prof. Steffen Bondorf and Prof. Enrico Vicario.

I also want to thank Anja, Anja, Eric, Jens, Markus, Matthias, Paul and Vlad for making me feel so much at home during my time at DISCO lab. It was really fun.

I extend my thanks to the administrative staff of both Pisa and Florence universities, for their precious help through the bureaucracy needed for my activities, and in particular to Simona Altamura and her quick and punctual responses. I also thank the Regione Toscana, for the Pegaso grant that funded my PhD.

Thank you all,
Raffaele

Abstract

Worst-case analysis of networked systems is gaining importance due to the emergence of safety-critical applications with real-time requirements in many engineering applications, such as factory automation within the Industry 4.0 paradigm, automated or tele-operated driving, coordinated unmanned aerial vehicles, etc. With all these distributed applications, ex-ante certification that the end-to-end network traversal time is always below a known maximum is required to guarantee safety for humans and property. Deterministic Network Calculus (DNC) is a well-known theory that uses (min,+) and (max,+) algebra to infer deterministic worst-case bounds on the delay and backlog of network, representing traffic as a function of time, and network elements (e.g., regulators, schedulers, links) as operations that modify said functions. However, in nontrivial cases, computation of DNC expressions is not viable without mature software support. Furthermore, some types of DNC expressions are well known to be hard to compute, however research in the algorithmic aspects is again impeded without a solid foundation based on extensible software on which improvements may be developed and tested. Existing software does not match the above criteria.

In the work presented in this thesis, we filled this gap developing *Nancy*, a computational library with rich support for DNC operations and designed with an extensible, layered software architecture that implements the state of the art of DNC algorithms, i.e., the framework of Ultimately Pseudo-Periodic functions. Moreover, the library uses software engineering techniques for efficient use of memory and the parallelism available in multicore systems. This architecture enables to specialize algorithms to integrate DNC results for performance improvements, as well as perform new research on the algorithmic aspects of DNC. We show examples of this by discussing our own research on the topic, where we focus on a few practical use cases and provide novel algorithms and optimizations that improve their computation times by orders of magnitude. We also discuss the functional and design differences between *Nancy* and *RTC Toolbox*, showing, aided by synthetic benchmarks, the benefits in stability caused by the use of rational numerical types over floating point. Lastly, as *Nancy* is released as an open-source software, it provides to the research community a solid base for future DNC research, from the implementation of new studies to similar endeavors on the algorithmic aspects.

Contents

Acronyms

CPL Concave/Convex Piecewise Linear curves 53

DNC Deterministic Network Calculus 9–14, 16, 18–20, 22, 23, 25–28, 32, 34, 51–56, 59, 67, 83, 89, 151, 193

DRR Deficit Round Robin 22

GPS Generalized Processor Sharing 22

IWRR Interleaved Weighted Round Robin 22, 23, 116–118

lcm least common multiple 36, 53, 55, 135, 187–189

MPA Modular Performance Analysis 185

NC Network Calculus 14, 15, 51, 117

RTC Real-Time Calculus 23, 26, 52, 53, 185

SAC subadditive closure 17, 20, 21, 39, 40, 54, 58, 59, 122, 130, 133

SNC Stochastic Network Calculus 14, 51, 54

UA Ultimately Affine 27, 28, 55, 75, 82, 109, 117, 118, 123–125, 129

UC Ultimately Constant 2, 28, 96–99, 105–107, 109, 226, 233

UI Ultimately Infinite 2, 28, 32–34, 38, 39, 41, 75, 96, 98, 99, 105–107, 109, 207–209, 212, 226, 233

UPP Ultimately Pseudo-Periodic 9–11, 25–28, 31, 32, 38–40, 42, 52, 53, 55, 56, 58–60, 63, 73, 75, 88, 91, 95, 96, 105, 109, 117–119, 151, 152, 161, 163, 168, 173, 178, 186, 187, 193, 208, 211, 212, 237, 238, 253, 255, 259, 262

VCC Variability Characterization Curves 186, 187, 189

WRR Weighted Round Robin 22

Part I

Introduction and Background

Chapter 1

Introduction and Motivation

Worst-case analysis of networked systems is gaining importance due to the emergence of safety-critical applications with real-time requirements in many engineering applications, such as factory automation within the Industry 4.0 paradigm, automated or tele-operated driving, coordinated unmanned aerial vehicles, etc. With all these distributed applications, ex-ante certification that the end-to-end network traversal time is always below a known maximum is required to guarantee safety for humans and property. DNC [Cru91a; Cru91b; Cha00; LT01] is a well-known theory that uses (min,+) and (max,+) algebra [BCOQ92] to infer deterministic worst-case bounds on the delay and backlog of network traffic. It represents traffic as a function of time, and network elements (e.g., regulators, schedulers, links) as operations that modify said functions. Given bounds on the input of a traffic flow (e.g., as enforced by a traffic shaper at the entrance of a network) and knowledge of minimum service given to that flow at network elements (e.g., as enforced by possibly different per-flow schedulers at every hop), DNC allows one to compute the maximum delay that traffic from that flow will undergo, under any possible scenario.

Algebraically speaking, DNC relies on few basic operations, such as minimum, (min,+) convolution and subadditive closure, which can be composed in arbitrary sequences or nested into one another. However, in less-than-trivial cases, pen-and-paper computation of DNC expression is not viable, and automated computation of the relevant performance measures (e.g., a maximum delay for a flow traversing a multi-hop network) is instead required. Defining efficient computational representations for functions of time and algorithms implementing DNC operations thereupon is by no means a simple task. Works [BT08; BBL18] discuss the above issue at length, identifying UPP functions as the most general class closed with respect to DNC operations. These are piecewise affine functions, that have an initial "transient" part, followed by a "period" which is repeated an infinite amount of times. UPP curves arise spontaneously in several practical cases: work [BD22] shows that they do whenever packetization and finite transmission speeds are factored in. Moreover, flow-controlled networks, currently being envisaged for data centers [GSSAA19; WCHLL21; GC21], have UPP service curves, and so do wormhole-routing networks [QLD10] used in systems architectures.

The complexity of some DNC operations, such as (min,+) convolution, is superquadratic with respect to the number of linear pieces of the operands [BT08, Table 1]. Others DNC operations, such as the subadditive closure, have exponential complexity [BT07, p. 41]. This applies to both the number of elementary operations involved and the number of linear pieces of the result. Thus, it presents a challenge for the practical use of DNC analysis, as chaining multiple DNC operations can lead to expressions that are visually and algebraically neat, but computationally infeasible, e.g., they may take days to compute, or exhaust the hardware resources and not compute at all. This highlights the need for efficient software libraries that can effectively implement DNC operations.

To the best of our knowledge, there are no public, open-source *libraries* that implement DNC operations working on UPP curves. By "library", we mean a set of implementations of (min,+) and (max,+) (henceforth, we will use (•,+) when referring to both) algebra operations, providing ease of constructing and manipulating curves, and allowing a user to specialize algorithms for improved efficiency by adding new code paths or subclasses. The two existing libraries that can handle UPP curves are the *RTC Toolbox* [WTa] and the *RTaW-Pegase* library [RTaWc]. The former is a publicly available Java library, whose source code is not publicly available. Thus, one cannot improve on its bugs (e.g., it sometimes cycles indefinitely without any discernible cause [ZSa]) or lack of features, such as the lack of subadditive closure or of a function to find the intersection between two curves (which is essential to speed up several algorithms). *RTaW-Pegase* is proprietary, and its license does not allow it to be used for benchmarking or verification purposes. An online interpreter is freely available to try its functionalities [RTaWb], but its use is limited by the browser interface and license. To the best of our knowledge, the *COINC* library [BCGHLL09] is no longer available.

There are, instead, several DNC *network analysis tools* available, which are software packages that implement methods for analyzing specific types of networks. These often implement their own versions of (•,+) algebra operations, which are typically restricted to classes of curves that match the type of network they are designed to analyze. Many are mentioned in [BBL18], while a review of their capabilities is reported in [ZHLC20]. For example, *DEBORAH* [BLMS10] analyzes FIFO tandems of rate-latency curves traversed by leaky-bucket-shaped flows, and therefore it only implements (min,+) convolution and subtraction of pseudo-affine curves, which are algorithmically trivial. The *NC-TANDEM-TIGHT* tool [BJT10; BT16] analyzes arbitrary multiplexing tandems by modeling their worst-case delay computation as a linear programming problem, which implicitly rules out generic

UPP curves, which would yield non-convex programs instead. *NC-Maude* [Boy10] is a tool written in Maude, a high-performance reflective language. It uses a rational numeric type instead of floating point, however it supports only simple functions such as rate-latency and token-bucket as inputs. It was released as an open and extensible tool, though it appears to be no longer available. The *DiscoDNC* tool [BS14], now *NCorg DNC*, is limited to ultimately concave/convex piecewise affine curves, for which DNC operations are considerably simpler. The work [LHL17a] describes a tool written in the NVIDIA CUDA language that computes convolutions and deconvolutions using GPUs, claiming improved efficiency over standard CPU-based computation. However, there seems to be no executable or code available to go with this paper. Unfortunately, none of these tools provide reusable, extensible, general-purpose DNC operations. Without such a library, it is difficult to advance new research in the algorithmic aspects of Deterministic Network Calculus, as one is limited by the capabilities and exposed APIs of each tool.

We filled the gap above by designing and developing *Nancy* [ZS22], an open-source DNC library that works with arbitrary UPP curves. *Nancy* is coded in C#, consists of more than 32k lines of code, and is natively parallel. It provides a rich API with many properties and methods to cover the needs of most DNC research and applications, which can be easily broadened thanks to its open-source MIT license and its extensible architecture. It includes a thorough documentation – both in-code, visible through an IDE, and online [ZSb] – which provides both explanations on each method and literature references. Furthermore, it is well integrated in the .NET ecosystem, providing packages via NuGet [ZSc], so that it can be easily added and used in new projects. Its source code is hosted on GitHub [ZSd], and it uses CI/CD pipelines to keep the public releases up to date with the code.

These properties were invaluable in our own research, as we were able – whenever we found issues, unexpected results or computationally taxing operations – to use the open nature of *Nancy* to investigate them, and further adapt the tool to our needs. This led us to develop novel algorithmic extensions and optimizations, covering a larger set of operations and properties used in DNC and improving upon the state of the art to make more studies feasible to run on modern hardware, results that – we believe – would have not been achieved without developing *Nancy* first.

In this thesis, we discuss the above results, navigating between the two natures of them: on one hand, we will discuss the mathematical models, theorems and properties – both from literature and from our novel research – and on the other hand, we will discuss the software engineering challenges and solutions involved in the development of these results into an accessible library that makes efficient use of memory and the parallelism available in multicore systems. As such, we will not shy away from discussing implementation details and techniques whenever we believe they are relevant w.r.t. the observable properties of the *Nancy* library as an ongoing, actively maintained project.

We show how, considering a few practical use cases, our optimizations improve their computation times by orders of magnitude, turning tens of minutes into less than a second.

We also compare some of the functional features and design choices of *Nancy* and *RTC Toolbox*, discussing the choice of floating point numerical types and showing, aided by synthetic benchmarks, how it leads to instable code.

This thesis is organized as follows. In Part I, which includes this introduction, we present the existing work and literature background. In Chapter 2, we provide a small introduction to Deterministic Network Calculus, and some applications which will be useful to contextualize our contributions. In Chapter 3, we detail the mathematical model used throughout this thesis. In Chapter 4, we mention a collection of formal results from the literature, adapted to our mathematical model, that we are going to use within this thesis. In Chapter 5, we discuss the Related Works. In Chapter 6, we provide a research statement, summarizing the context and objectives of our work.

In Part II, we present the novel contributions. In Chapter 7, we present an overview of the contribution. In Chapter 8, we present the *Nancy* library and its architecture, discussing in detail some relevant methods. In Chapter 9, we discuss some useful algorithms that are implemented in *Nancy*, which emphasize the benefits of a rich API for practical uses. In Chapters 10 and 11, we present extensions to the toolbox to include, respectively, pseudoinverses and composition. In Chapter 12, we present the *representation minimization* algorithm, and show its benefits for chained computations. In Chapter 13, we present the improved (min,+) convolution algorithms for subadditive curves, and show how they dramatically improve the computation time in use cases that involve them. In Chapter 14, we present the improved $(\cdot,+)$ convolution algorithms based on the isomorphism between the two. In Chapter 15, we discuss the functional and design differences between *Nancy* and *RTC Toolbox*, with a focus on the choice of numerical type. Finally, Chapter 16 concludes the work.

Chapter 2

Fundamentals and Use Cases of Deterministic Network Calculus

In this chapter, we discuss the fundamentals of the DNC framework, together with a selection of use cases which will serve as useful reference to discuss the context and impact of our contribution.

2.1 Fundamentals of Deterministic Network Calculus

A DNC flow is represented as a non-decreasing, non-negative and left-continuous cumulative function $A : \mathbb{R}_+ \to \mathbb{R}_+$. Function $A(t)$ represents the number of bits of the flow observed in $[0, t[$. In particular, $A(0) = 0$. Flows can be constrained by *arrival curves*. A non-decreasing, non-negative and left-continuous function α is an *arrival curve* for a flow A if:

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

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

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

as shown in Figure 2.1. This means, among other things, that the long-term arrival rate of the flow cannot exceed ρ.

Let A and D be the functions that describe the same data flow at the input and output of a lossless network element (or *node*), respectively. If that node does not create data internally (which is often the case), causality requires that $A \geq D$. We say that the node behavior can be modeled via a *service curve* β if

$$\forall t \geq 0, \quad D(t) \geq \inf_{0 \leq s \leq t} \left\{ A(s) + \beta(t - s) \right\}. \tag{2.1}$$

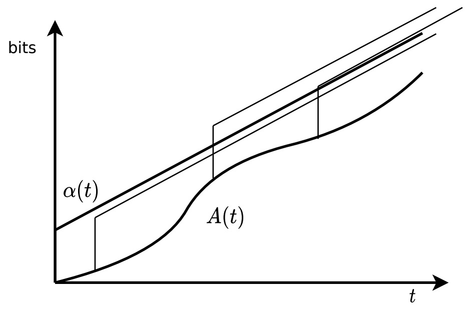


Figure 2.1: Example of leaky-bucket shaper. The traffic process $A(t)$ is always below the arrival curve $\alpha(t)$ and its translations along $A(t)$.

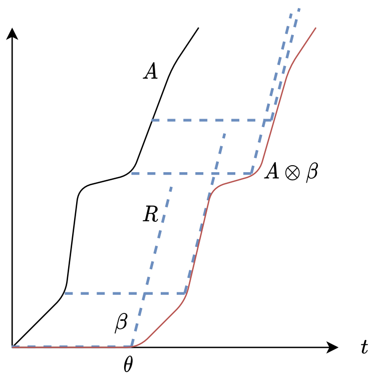


Figure 2.2: Graphical interpretation of the convolution operation. A is the input function, $\beta_{R,\theta}$ is a rate-latency service curve, and $A \otimes \beta$ is a lower bound on the output.

In that case, the flow is guaranteed the (minimum) service curve β – which is often assumed to be non-decreasing, non-negative and left-continuous. The infimum on the right-hand side of Equation (2.1), as a function of t, is called the *(min,+) convolution* of $A(t)$ and $\beta(t)$, and is denoted by $(A \otimes \beta)(t)$. The dependency on t is omitted whenever it is clear from the context. The alert reader can check that convolution is commutative and associative. Computing the above convolution entails sliding β along A and taking the *lower envelope* of the result (i.e., the infimum for each time instant). Note that, due to the close relationship between *functions* (the mathematical object) and *curves* (the DNC model), in the following we may use both interchangeably.

One key strength of NC[1] , called *scheduling abstraction* in [CS12], is that the complexities of several network elements, such as delay elements, schedulers or links, can be modeled and abstracted away via suitable service curves. A very frequent case is the one of *rate-latency* service curves, defined as

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

[1]This includes also the *stochastic* counterpart of DNC, Stochastic Network Calculus (SNC).

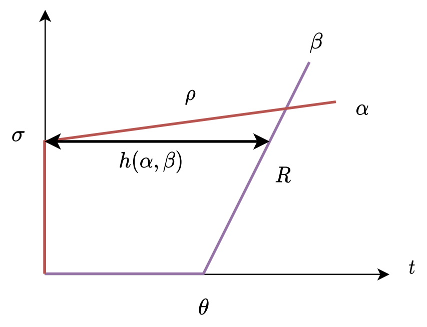


Figure 2.3: Graphical example of a delay bound.

for some $\theta \geq 0$ (the latency) and $R > 0$ (the rate). Notation $[.]^+$ denotes $\max(., 0)$. For instance, a constant-rate server (e.g., a wired link) can be modeled as a rate-latency curve with zero latency. Figure 2.2 shows the lower bound of D obtained by computing $A \otimes \beta$, with $\beta = \beta_{R,\theta}$. In the case of scheduling elements, we distinguish *per aggregate* and *per-flow* service curves, i.e., whether the service is provided to all flows or specifically to one. The latter is also called *leftover* service curve.

Another point of strength of NC, called *convolution-form networks* in [CS12], is that service curves models are *composable* via the (min,+) convolution: the end-to-end service curve of a tandem of nodes traversed by the same flow can be computed as the convolution of the service curves of each node.

For a flow that traverses an element with a known service curve (be it a single node, or the end-to-end service curve of a tandem, computed as discussed above), a tight[2] *upper bound* on the delay can be computed by combining its arrival curve α and the service curve β itself, as follows

$$h(\alpha, \beta) = \sup_{t \geq 0} \left\{ \inf \left\{ d \geq 0 \mid \alpha(t - d) \leq \beta(t) \right\} \right\}. \tag{2.2}$$

The quantity $h(\alpha, \beta)$ is known as the maximum horizontal distance between α and β, as shown in Figure 2.3. 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 uses results which appear simple, and can often be computed with pen and paper. However, in many practical cases, the computations become more contrived, requiring software automation and efficient algorithms. We discuss here some of these use cases, which we will reference later on to contextualize our contributions and their impact.

[2]By *tight* we mean that no better (i.e., smaller) upper bound can be derived given α and β – if the latter are not tight w.r.t. the entities they model, of course the bound will not be either.

2.2 (min,+) and (max,+) algebra

In the previous section, we introduced the (min,+) convolution, both to bound the departure process of a flow traversing a server, and an operator to compose service curves of nodes in a tandem to obtain a service curve for the tandem as a whole. This operator derives from the (min,+) algebra, which is a dioid where the addition is replaced with the minimum and the product with the sum.[3] The main operations between functions in this algebra are

- The minimum, $f \wedge g$,
- The addition, $f + g$,
- The (min,+) convolution, $f \otimes g = \inf_{0 \leq s \leq t} \{f(s) + g(t-s)\}$,
- The (min,+) deconvolution, $f \oslash g = \sup_{s \geq 0} \{f(t+s) - g(s)\}$.

We also mention the (max,+) algebra, which is defined similarly to (min,+) although replacing minimum with the maximum. Thus, the main operations are

- The maximum, $f \vee g$,
- The addition, $f + g$,
- The (max,+) convolution, $f \overline{\otimes} g = \sup_{0 \leq s \leq t} \{f(s) + g(t-s)\}$,
- The (max,+) deconvolution, $f \overline{\oslash} g = \inf_{s \geq 0} \{f(t+s) - g(s)\}$.

We refer to books [Cha00; LT01; BBL18] for a more complete presentation of the algebraic foundations of DNC, and how these operators map to valuable insights in network analysis using DNC. For a more complete presentation of (min,+) and (max,+) algebra, both refer to the pioneering work of [BCOQ92]. To the purposes of this thesis, we limit the discussion to results that we use as motivating examples.

(max,+) algebra can be used equivalently to (min,+) for system modeling, if one swaps the usual meaning of the axes, i.e., *data* for the x-axis, and *time* for the y-axis, and functions are assumed to be right-continuous instead of left-continuous.

In [Lie17] it is shown that one can switch from a model to the other using lower and upper pseudoinverses. While a formal definition is provided later on in Section 4.3, pseudoinverses can be explained by observing that the non-decreasing property does not prevent a function f to be constant on an interval $]a, b[$, hence it may not be bijective. Thus, an inverse f^{-1} may not be defined. Pseudo-inverses address this issue by "deciding" what $f^{-1}(x)$, for $x \in]a, b[$, should be equal to: for

[3]While commonly called *algebras* in DNC jargon, (min,+) and (max,+) are *semirings* [BBL18, Ch. 2].

the lower, $f_{\downarrow}^{-1}(x) = a$, while for the upper $f_{\uparrow}^{-1}(x) = b$. Furthermore, the $f_{\downarrow}^{-1}$ is left-continuous, while $f_{\uparrow}^{-1}$ is right-continuous.

What is shown in [Lie17], then, is that one can switch from a (min,+) model f to its (max,+) equivalent by computing $f_{\uparrow}^{-1}$; conversely one can switch from a (max,+) model f to its (min,+) equivalent by computing $f_{\downarrow}^{-1}$.

Another example of expression using pseudoinverses is in [LT01, p. 128], which shows that the horizontal deviation can be computed as

$$h(\alpha, \beta) = \sup_{t \geq 0} \left\{ \beta_{\downarrow}^{-1}(\alpha(t)) - t \right\}. \tag{2.3}$$

Lastly, we introduce the subadditive closure and their corresponding (max,+) operator, the *superadditive closure*.

Definition 2.1 (Subadditive Closure). The *subadditive closure* (*SAC*) of a non-decreasing function f is defined as

$$\overline{f(t)} = \inf_{n \geq 0} \left\{ f^{(n)}(t) \right\}, \tag{2.4}$$

where $f^{(n)}$ denotes the n-fold (min,+) convolution of f with itself, i.e., $f^{(0)} = \delta_0$, $f^{(1)} = f$, and $f^{(n)} = \bigotimes_{i=1}^{n} f$ for $n \geq 1$. Function δ_0 is an infinite step in $t = 0^+$, i.e., $\delta_0(0) = 0, \delta_0(t) = +\infty, \forall t > 0$.

Note that $\overline{f}$ is subadditive, as the name suggests. The formal definition of subadditivity is the following.

Definition 2.2 (Subadditive Function). f is subadditive if and only if $f(u) + f(s) \geq f(u+s)\ \forall u, s$.

Moreover, given a function f such that $f(0) = 0$, if f is subadditive then $\overline{f} = f$. Otherwise, it is $\overline{f} \leq f$. Convolution does preserve subadditivity, as per the following property.

Theorem 2.3 (Convolution of Subadditive Functions [LT01, Theorem 3.1.9]). *If f and g are subadditive functions, so is $f \otimes g$.*

Moreover, the SAC of a minimum is the convolution of the SACs of the operands, i.e.,

Theorem 2.4 (SAC of a Minimum [LT01, Theorem 3.1.11]). $\overline{f \wedge g} = \overline{f} \otimes \overline{g}$

Note that the subadditive closure of a generic curve is a very complex operation: as we discuss later in Section 3.6, it is $\mathcal{NP}$-hard. An exception is the case of $\overline{\beta + W}$, where β is a rate-latency curve and W is a function f such that $f(0) = 0$ and $f(t) = W$ for all $t > 0$. For this case, [LT01, pp. 118-9] shows it can be computed in closed form.

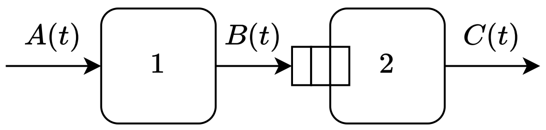


Figure 2.4: Network with static window flow-control.

The corresponding concepts in (max,+) algebra are *superadditive closure* and *superadditivity*.

Definition 2.5 (Superadditive Closure)**.** The *superadditive closure* of a non-decreasing function f is defined as

$$\overline{\overline{f(t)}} = \sup_{n \geq 0} \left\{ f^{\overline{(n)}}(t) \right\}, \tag{2.5}$$

where $f^{\overline{(n)}}$ denotes the n-fold (max,+) convolution of f with itself, i.e., $f^{\overline{(0)}} = \overline{\delta}_0$, $f^{\overline{(1)}} = f$, and $f^{\overline{(n)}} = \overline{\otimes}_{i=1}^{n} f$ for $n \geq 1$. Function $\overline{\delta}_0$ is a minus infinite step in $t = 0^+$, i.e., $\overline{\delta}_0(0) = 0, \overline{\delta}_0(t) = -\infty, \forall t > 0$.

Note that $\overline{\overline{f}}$ is superadditive, as the name suggests. The formal definition of superadditivity is the following.

Definition 2.6 (Superadditive Function)**.** f is superadditive if and only if $f(u) + f(s) \leq f(u+s)\ \forall u, s$.

In the next subsections we discuss two use-cases of DNC. They present specific problems regarding algorithms and their computation efficiency, that will serve as a basis for the contributions of this thesis.

2.3 Networks with flow control

DNC can be used to model network elements having flow control [LT01, Chapter 4]. Flow control finds application in wormhole networks-on-chip [QLD09; GM18; GM19], can be used to model manufacturing systems [BJLL06], and are currently being envisaged for data centers [GSSAA19; WCHLL21; GC21]. Consider the network in Figure 2.4, in which a flow traverses nodes 1 and 2, which have static flow control due to the limited buffer in 2. Let β_1, β_2 be the service curves offered by nodes 1 and 2 to the flow that we are observing. Node 1 will thus serve that flow's traffic, with service curve β_1, only if there is already buffer space available in 2; in turn, the available part of this buffer space, whose size is $W > 0$, depends on the ability of 2 to serve that flow's traffic with its own service curve β_2. We also assume that 1 is instantaneously aware of the current state of the buffer of 2. Thus, we can model the network as in Figure 2.5.

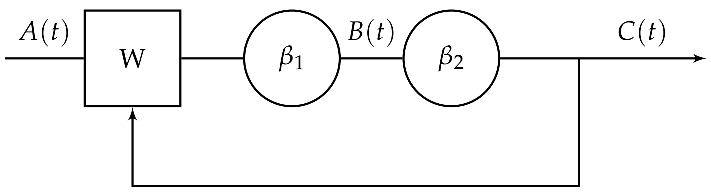


Figure 2.5: NC model for network with static window flow-control. Circles represent service curve elements, whereas rectangles represent flow control windows.

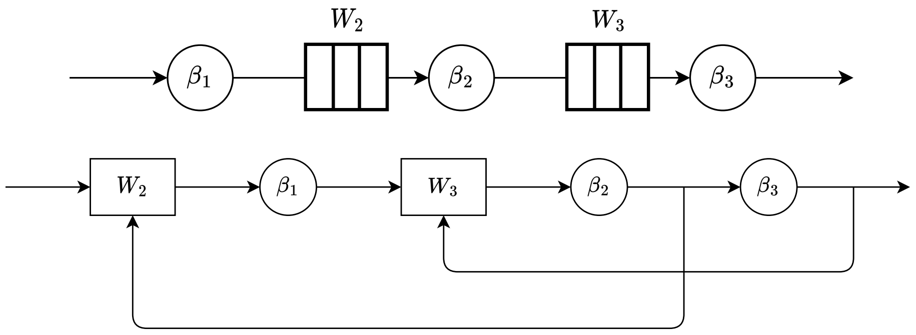


Figure 2.6: A tandem of three flow-controlled nodes and its DNC model.

In order to compute an end-to-end service curve for a flow traversing the above system, we must first get rid of the feedback arc in the NC model, transforming it into a tandem. This is done by computing first the *equivalent service curve* of node 1, β_1^{eq}, such that

$$B(t) \geq (A \otimes \beta_1^{eq})(t).$$

The latter takes into account the reduction in the service brought on by the presence of the subsequent flow control. It is [LT01, Chapter 4]

$$\begin{aligned} \beta_1^{eq} &= \beta_1 \otimes \beta_{fc}, \\ \beta_{fc} &= \overline{\beta_1 \otimes \beta_2 + W}. \end{aligned}$$

Then, the system offers to the flow an end-to-end service curve β^{eq}, so that

$$C(t) \geq (A \otimes \beta^{eq})(t).$$

β^{eq} is equal to

$$\begin{aligned} \beta^{eq} &= \beta_1^{eq} \otimes \beta_2 \\ &= \beta_1 \otimes \beta_2 \otimes \beta_{fc}. \end{aligned} \tag{2.6}$$

The extension of the above method to tandems of size n is straightforward: consider for instance the tandem in Section 2.3. Nodes 2 and 3 have limited buffers of

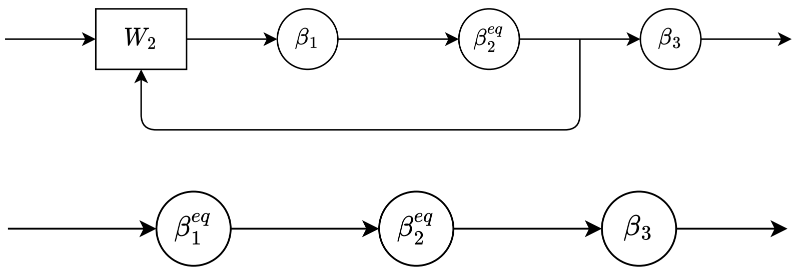


Figure 2.7: Transformation of the DNC model to an equivalent tandem.

size W_2 and W_3. The resulting DNC model is shown in the figure. To find the end-to-end service curve of the system, β^{eq}, we iterate the above methodology – starting from the rightmost node – and compute the following

$$\begin{aligned}
\beta_2^{eq} &= \beta_2 \otimes (\overline{\beta_2 \otimes \beta_3 + W_3}),\\
\beta_1^{eq} &= \beta_1 \otimes (\overline{\beta_1 \otimes \beta_2^{eq} + W_2}),\\
\beta^{eq} &= \beta_1^{eq} \otimes \beta_2^{eq} \otimes \beta_3.
\end{aligned}$$

The above method is also illustrated in Figure 2.7. By expanding the expression of β_1^{eq}, we obtain

$$\beta_1^{eq} = \beta_1 \otimes (\overline{\beta_1 \otimes \beta_2 \otimes (\overline{\beta_2 \otimes \beta_3 + W_3}) + W_2}). \tag{2.7}$$

Equation (2.7) includes a nested SAC. As we will justify in the following chapters, this implies that – even in the simplest cases – this involves $\mathcal{NP}$-hard computations.

This method can be generalized to a tandem of n nodes, as:

$$\begin{aligned}
\beta_{n-1}^{eq} &= \beta_{n-1} \otimes \overline{\beta_{n-1} \otimes \beta_n + W_n},\\
\beta_i^{eq} &= \beta_i \otimes \overline{\beta_i \otimes \beta_{i+1}^{eq} + W_{i+1}},\\
\beta^{eq} &= \left(\bigotimes_{i=1\ldots n-1} \beta_i^{eq} \right) \otimes \beta_n.
\end{aligned} \tag{2.8}$$

This method of analysis (henceforth: the *exact method*) therefore requires $\mathcal{O}(n)$ nested SACs for a tandem of n nodes. All (except possibly the first) are nontrivial to compute (see Section 3.6). In practice, despite the apparent conciseness of (2.8), computing β^{eq} via this method is computationally infeasible.

In [BJLL06], a property was proved that lower bounds β_i^{eq} with a *convolution of SACs*:

$$\beta_i^{eq'} = \beta_i \bigotimes_{j=i}^{n-1} (\overline{\beta_j \otimes \beta_{j+1} + W_{j+1}}). \tag{2.9}$$

Then, an end-to-end service curve can be computed as:

$$\begin{aligned}
\beta^{eq'} &= \bigotimes_{i=1}^{n-1} \beta_i^{eq'} \otimes \beta_n \\
&= \beta_1^{eq'} \otimes \beta_2^{eq'} \otimes \cdots \otimes \beta_{n-1}^{eq'} \otimes \beta_n \\
&= \left(\beta_1 \otimes \bigotimes_{i=1 \ldots n-1} \overline{\beta_i \otimes \beta_{i+1} + W_{i+1}} \right) \\
&\quad \otimes \left(\beta_2 \otimes \bigotimes_{i=2 \ldots n-1} \overline{\beta_i \otimes \beta_{i+1} + W_{i+1}} \right) \otimes \cdots \otimes \beta_n \\
&= \left(\bigotimes_{i=1 \ldots n} \beta_i \right) \otimes \bigotimes_{i=1 \ldots n-1} \overline{\beta_i \otimes \beta_{i+1} + W_{i+1}}. \qquad (2.10)
\end{aligned}$$

The above is a consequence of each $\overline{\beta_i \otimes \beta_{i+1} + W_{i+1}}$ being subadditive, thus $f \otimes f = f$.
Authors of [BJLL06] prove that:

$$\beta_i^{eq} \geq \beta_i^{eq'} \; \forall i = 1 \ldots n-1. \qquad (2.11)$$

From the above, since convolution is isotonic, it follows that:

$$\beta^{eq} \geq \beta^{eq'}. \qquad (2.12)$$

Computing $\beta^{eq'}$ via (2.10) (henceforth: the *approximate method*) is computationally more tractable – if all the service curves β_i are rate-latency – because it does away with nested SACs. However, it still requires one to compute $\mathcal{O}(n)$ *convolutions* of nontrivial subadditive curves.

An exact expression for the service curve of the *first* node in a tandem of flow-controlled nodes has been derived in [BPC09]. Its computation requires a chain of convolutions of subadditive curves, i.e., the same type which we encounter here and whose computation we will optimize later in this thesis.

2.4 Round-robin schedulers in DNC

As another example, we discuss some results of DNC for Round Robin schedulers, and in particular the models for IWRR and DRR based on (min,+) algebra expressions.

Round Robin schedulers are packet-based implementation of the ideal GPS policy. They are used in packet-switched networks and real-time processing systems to handle resource allocation between multiple flows (or tasks), and are, roughly speaking, based on the idea of letting each use the service in turn within each *round*. WRR assigns, as the name suggests, a *weight* w_i to each flow f_i, and gives to each flow, in each round, the opportunity to send w_i packets *consecutively* – therefore, if every flow has data to send, we have an average per-flow rate of $w_i / \sum_{j \neq i} w_j$. However, due to the rotation between waiting and burst transmission, the burstiness of each flow may be increased. IWRR avoids this issue by, as the name suggests, *interleaving* the transmission of packets from the different flows, their overall allocation per-round remaining unchanged. DRR deals with scenarios where some flows may only transmit infrequently, and provide to those flows more service when this occurs: the algorithm uses in fact a *quantum* that, like the *weight* in WRR, is used to limit how much a flow can transmit each round – however, the quantum is preserved from a round to the next, acting as a *deficit* term that will provide more service to the flow when it will have data to transmit. Moreover, while WRR counts packets, DRR counts bits.

Several works in DNC literature provide models to derive leftover service curves for these schedulers. [BBL18] provides three different curves for WRR, based on different assumptions and computational complexity, while [TLB21] provides a strict leftover service curve for IWRR. Work [CNS22a; CNS22b] shows how the constraints on contending flows can be taken into account to improve on these results for WRR and IWRR. For DRR, [BSS12a] and, lately, [Bou21] and [TB21; TL22] provide strict per-flow service curves.

What we highlight, for this thesis, are the operations and operands that some of these studies consider. [BBL18, Theorem 8.6] uses, to compute the per-flow service curve for a WRR scheduler, lower pseudoinverse and composition; in [TLB21, Theorem 1], the result for IWRR scheduler is computed, again, using lower pseudoinverse and composition; in [TL22], authors show an iterative process using pseudoinverse and non-decreasing closure to iteratively improve a strict per-flow service curve for DRR.

These results therefore highlight the need of software support for the aforementioned operators. While the algebraic formulation of these three operations is well known, their algorithmic aspects have not been addressed, to the best of our knowledge – contrary to, e.g., (min,+) convolution, as we show in the following chapter.

Furthermore, for many of these studies the resulting curve is staircase shaped – even if the underlying per aggregate service curve is a rate-latency one. Thus, to use these results effectively, one needs software support for (min,+) convolution and horizontal deviation for such nontrivial curves.

As a reference example, we will consider [TLB21, Theorem 1], which we report here slightly rephrased, and show in Part II how it can be effectively translated into code using *Nancy*.

Theorem 2.7 (Strict Per-Flow Service Curves for IWRR)**.** *Assume n flows arriving at a server performing Interleaved Weighted Round Robin with weights $w_1, \dots, w_n$. Let $l_i^{\min}$ and $l_j^{\max}$ denote the minimum and maximum packet size of the respective flow. Let this server offer a superadditive 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

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

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

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

2.5 Real-Time Calculus

RTC is a mathematical framework for the worst-case analysis of real-time systems. It is based off DNC, and maintains many similarities with it – the differences are in fact related to the systems being modeled, composed of processors and tasks, while the mathematical foundations are the same [TCN00; Wan06], and a formal link between the two is provided in [BJT09]. In fact, RTC uses (min,+) and (max,+) algebra models and operations.[4]

Thus, the results discussed in this thesis, as well as the *Nancy* library, although they have roots in DNC, can be also used for and applied to RTC.

[4]Indeed, both the two other libraries we mention have *Real Time* in their name.

Chapter 3

Mathematical Model

3.1 Foreword

The DNC literature is rather inhomogeneous in its choice of mathematical model. For example, [LT01, p. 105] focuses on the set $\mathcal{F}$, which denotes the set of non-decreasing sequences ($t \in \mathbb{Z}$) or functions ($t \in \mathbb{R}$) such that $f(t) = 0$ for $t < 0$, and whose range is $[0, +\infty[$. In [BBL18, p. 19], instead, $\mathcal{F}$ denotes the functions defined in $\mathbb{R}_+ \to \mathbb{R} \cup \{+\infty\}$. In both works, only the *lower* pseudoinverse is defined and denoted as f^{-1}. Then, in [Lie17], both lower and upper pseudoinverses are defined, and $\mathcal{F}$ denotes the set of left-continuous, non-negative and non-decreasing functions in $\mathbb{R} \to \mathbb{R}_+ \cup \{+\infty\}$. In each of these works, as well as the others we reference, the choice is justified as the best fit to address the contribution of the work while excluding uninteresting edge cases. For similar reasons, in this thesis we introduce yet another set of functions, $\mathcal{U}$, and choice of notation. While the formal definitions, with examples, follow below, we here wish to highlight the key points in which our model differs from the above works (and others), and justify why.

First, we consider $\mathbb{Q}$ instead of $\mathbb{R}$, for two reasons, which both relate to the fact that we aim for a mathematical model that is *implementable*, i.e., that can be translated in software without introducing formal issues that are not addressed by the model itself. First, it is shown in [BT08] that the set of UPP functions in $\mathbb{R}_+ \to \mathbb{R}$ is not closed on all operations, i.e., that are edge cases where the result is not a UPP function again – we have this property, instead, for functions in $\mathbb{Q}_+ \to \mathbb{Q}$ (with some caveats, discussed in Section 8.4.1). This property is fundamental from a software perspective, as it implies that once we design a type to represent such functions, we can use this as result type of all implemented operations. The second reason is that, while they are undoubtedly useful for their formal properties, numbers in $\mathbb{R} \setminus \mathbb{Q}$ cannot have a finite representation, hence could never be faithfully represented in software. While floating point numbers are broadly used to *approximate* numbers in $\mathbb{R}$, the error introduced by such approximation remains something to account for, as one otherwise risks incurring in either a) compute results that are incorrect by more than an acceptable threshold, or b) to incur in cases where an algorithm attempts to obtain, iteratively, an unrepresentable number, hence never terminating. We will come back on this point in Chapter 15, where we compare our solution to the *RTC Toolbox*, which uses floating point numbers.

A second key difference is that we do not generally assume functions to be non-decreasing or non-negative. As discussed in Chapter 2, the above is a sound assumption for any system model, since cumulative functions are indeed non-decreasing and non-negative. However, as shown in [BT08], to define algorithms for $(\cdot,+)$ operations it is often mandatory to decompose a function into multiple parts (e.g., its *transient* and *periodic* parts), which may not be non-decreasing. As we show in our contributions, such as in Chapter 13 and Chapter 14, it is advantageous to have such parts to be again functions in $\mathcal{U}$, as this allows us to apply algorithmic improvements also to parts of other algorithms. Furthermore, we allow functions to take negative values so that our model does not inhibit replication of literature results that use them, e.g., Lemma 3.26.

Finally, in order to represent functions with a finite amount of memory, we follow the model of [BT08] and consider functions defined in $\mathbb{Q}_+$ and that are piecewise affine and Ultimately Pseudo-Periodic. As we wish to flexibly represent functions of both (min,+) and (max,+), we do not assume either left- or right-continuity, and the range of functions in $\mathcal{U}$ is $\mathbb{Q} \cup \{-\infty, +\infty\}$.

Such differences do not always come without pain: while for most results in DNC literature one can trace the same steps of the proof to obtain the equivalent result for $\mathcal{U}$, in some others we need to deal with new edge cases that alter them, as is the case with the lower and upper pseudoinverse. We will, in general, provide the results adapted to apply on functions of $\mathcal{U}$, with reference to the original literature, and provide also the adapted proofs whenever they differ significantly from the referenced literature.

In the rest of this chapter, we provide an overview of the model and results from the literature that (be in their original form or via an adaptation) we use in this thesis. We discuss, instead, the extensions to this model that constitute novel contributions later in Part II.

3.2 The UPP model

To implement DNC and RTC computations in software, one needs to provide finite representations of functions and well-formed algorithms for $(\cdot,+)$ operations. According to the widely accepted approach described in [BT08; BBL18], a sufficiently generic class of functions useful for DNC 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, with reference to [BT08, pp. 8-9].

Definition 3.1 (Ultimately Pseudo-Periodic Function)**.** 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[1]

$$f(t + k \cdot d_f) = f(t) + k \cdot c_f, \qquad \forall t \geq T_f, \forall k \in \mathbb{N}. \tag{3.1}$$

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*. We also say that f is UPP *from* T_f, and that is has (*pseudo-periodic*) *slope* $\rho_f := c_f / d_f$.

Definition 3.2 (Piecewise Affine Function)**.** 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 $t \in \left]a_i, a_{i+1}\right[$.

In [BT08], this class of functions is shown (with some caveats, discussed later in Section 3.8) to be stable w.r.t. all (min,+)- operations, while functions $\mathbb{R}_+ \to \mathbb{R} \cup \{+\infty, -\infty\}$ are not.[2] As the (max,+) operations can be computed via equivalences based on (min,+) ([BBL18, p. 33], reported here as Lemma 3.26), these stability properties extend also to (max,+) operations.

We remark that functions in $\mathcal{U}$ are not necessarily non-decreasing. While DNC functions are usually assumed to be so, in order to implement (min,+) 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. A subclass of piecewise affine UPP functions are Ultimately Affine (UA) functions.

Definition 3.3 (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

$$f(t) = f(T_f^a) + \rho_f \cdot \left(t - T_f^a\right), \qquad \forall t \geq T_f^a, \tag{3.2}$$

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 [BT08], but we prove their equivalence in Proposition G.1. UA functions are (obviously) UPP as

[1]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 $\mathbb{N}$.

[2]An alternative class of functions with such stability is $\mathbb{N} \to \mathbb{R} \cup \{+\infty, -\infty\}$, however this is only useful to model systems where time is discrete, and may not be closed under operators such as pseudoinverses.

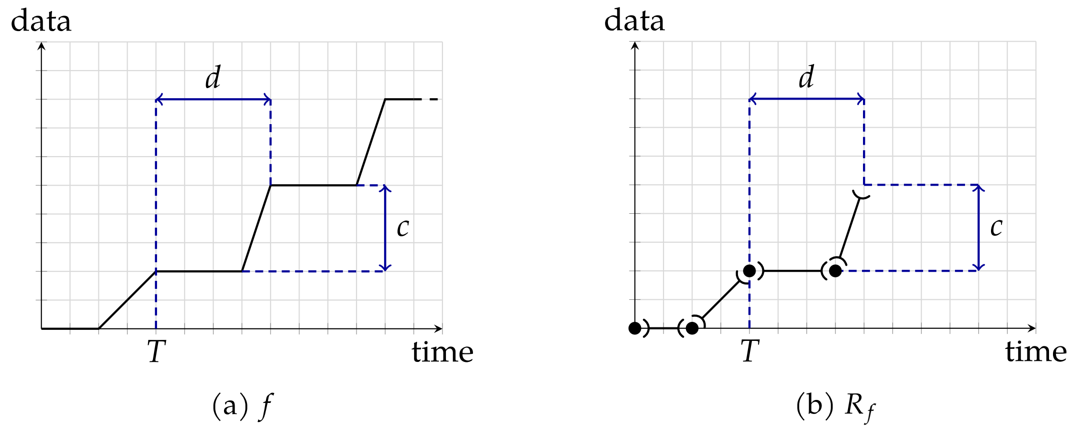


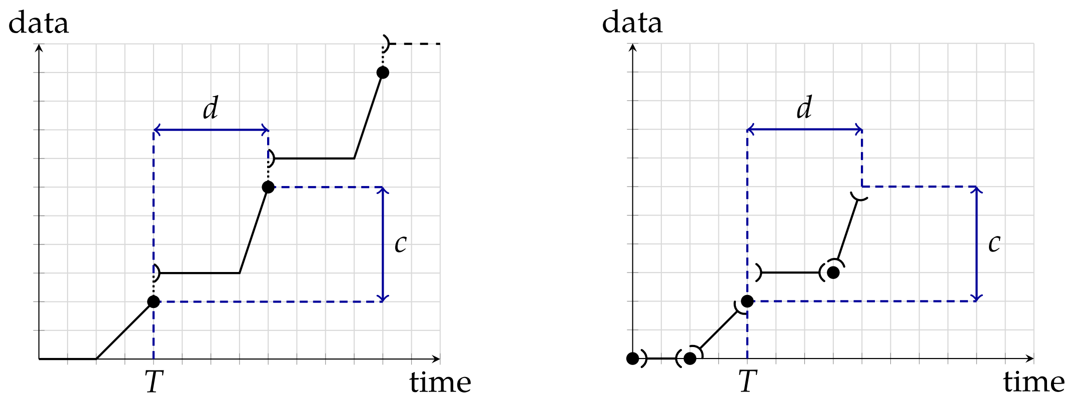
Figure 3.1: A continuous ultimately pseudo-periodic piecewise affine function f and its representation R_f.

well, their period being a single segment of slope ρ_f and arbitrary length starting at T^a_f. They have seen many applications in DNC, e.g., the arrival curve of a leaky-bucket shaper or a rate-latency service curve are both UA. A Ultimately Constant (UC) function is UA with $\rho_f = 0$. Similarly, a Ultimately Infinite (UI) function is UA with $f(t) = +\infty$, or $f(t) = -\infty$ for all $t \geq T^a_f$. Typical cases in DNC are the service curves of *delay elements*. Unlike UPP, the class of UA functions is not closed with respect to DNC operations. For instance, in Section 2.3 we show 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., [BSS12b; TLB21; TL22]). However, there are cases when simpler algorithms for DNC operations can be derived if one assumes that operands are UA. For this reason, there are DNC toolboxes that only consider UA functions, e.g., *NCorg DNC* [BS14].

Throughout this thesis, we will consider all functions to be in $\mathcal{U}$, hence, piecewise affine and UPP. When appropriate, we will impose that they are neither UC nor UI. The reason behind this assumption is that some properties do not hold unless we exclude these two subclasses. This limitation is of negligible practical impact, however, since $(\bullet,+)$ operations are trivial when operands are UC/UI functions, hence optimizations are hardly needed for these cases.

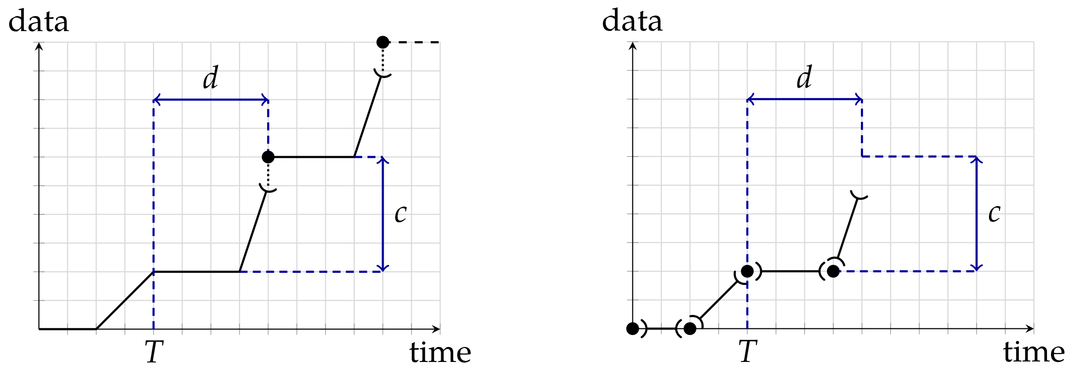
For functions in $\mathcal{U}$, it is enough to store a representation of the initial transient part, i.e., interval $\mathcal{T} = [0, T[$, and of one period, i.e., interval $\mathcal{P} = [T, T+d[$. This entails storing a description of the function over interval $[0, T+d[$, which is a finite amount of information. Figures 3.1 to 3.3 show examples of such functions.

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



(a) f (b) R_f

Figure 3.2: Example of a left-continuous ultimately pseudo-periodic piecewise affine function f and its representation R_f.



(a) f (b) R_f

Figure 3.3: Example of a right-continuous ultimately pseudo-periodic piecewise affine function f and its representation R_f.

easily model discontinuities. Moreover, we will use the umbrella term *elements* to encompass both when convenient.

Definition 3.4 (Point)**.** We define a *point* as a tuple

$$p_i := (t_i, f(t_i)), \qquad i \in \{1, \ldots, n\},$$

which describes f in t_i.

Definition 3.5 (Segment)**.** We define a *segment* as a tuple

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

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

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

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 3.6 (Sequence)**.** We define a sequence S_f^I as on ordered set of elements $e_1, \ldots, e_n$ that alternate between points and segments and describe f in the finite interval I.

Note that, given R_f, one can compute $f(t)$ for all $t \geq 0$, and also S_f^I, i.e., a sequence describing f in the finite interval I for any $I \subset \mathbb{Q}_0^+$. Furthermore, being finite, R_f can be used as a data structure to represent f in code. As is discussed in depth in Chapter 12, R_f is not unique, and using a *non-minimal* representation of f can affect the efficiency of the computations. Given a sequence S, let $n(S)$ be its cardinality.

In the following, we will call a sequence S_f^I, for a finite I that is, in general, $\neq \left[0, T_f + d_f\right[$, a *cut* of f. As we will show, cuts of functions, and their cardinalities, are a key part in the algorithms that implement $(\bullet,+)$ operations and their runtime. We define as well $Cut()$ to be an algorithm that, given R_f and an interval I, computes S_f^I. While its basic implementation is fairly obvious, we discuss in Section 8.5 how one can improve its active memory utilization.

With a little abuse of notation, we use $(\bullet,+)$ operators directly on cuts such as S_f^I. For instance, given the (min,+) convolution, we will write $S_f^{I_f} \otimes S_g^{I_g}$ to express that we are computing the (min,+) convolution $f \otimes g$, limited to the values of f in interval I_f and those of g in interval I_g.

Definition 3.7 (Breakpoint)**.** Let f be a function of $\mathcal{U}$. We say that t_b is a *breakpoint* of f if f is non-differentiable in t_b, i.e., either of the following is true:

- f has a discontinuity at t_b;
- the rates of f in t_b^- and t_b^+ are different.

It follows from the above definition that, for any breakpoint t_b internal to the support a sequence S_f, S_f *must* have a point – although it may also contain *unnecessary* points, thus with a larger cardinality $n\left(S_f\right)$ than necessary. We call a sequence *well-formed* if there are no such unnecessary points. We will pick this concept again in Chapter 12. It will be useful in the following to consider a function f of $\mathcal{U}$ in a *restricted support D*.[3] This is done as follows.

[3]Inspired by [BT08, p. 7], we use *support D* to assign a subset outside which the function is constantly $-\infty$ or $+\infty$. Note that this does not necessarily mean, in general, that f is finite on D. We mostly use it to define a set within which the properties of f are observed.

Definition 3.8 (Min and Max Restrictions)**.** Let f be a function of $\mathcal{U}$ and let $D \subseteq \mathbb{Q}$. Then, its *min restriction* over a support D is defined as

$$f|_D^\wedge := \begin{cases} f(t), & \text{if } t \in D, \\ +\infty, & \text{otherwise,} \end{cases}$$

Moreover, its *max restriction* over a support D is defined as

$$f|_D^\vee := \begin{cases} f(t), & \text{if } t \in D, \\ -\infty, & \text{otherwise.} \end{cases}$$

In this thesis, we will often consider D to be an interval I of the form $[0, a[$ or $[a, +\infty[$. The above definition also allows us to write $f|_t^\wedge$ and $f|_t^\vee$ for the transient part, i.e, if $I = \left[0, T_f\right[$, and $f|_p^\wedge$ and $f|_p^\vee$ for the respective pseudo-periodic part, i.e., $I = \left[T_f, +\infty\right[$. Henceforth, we will use the shorthand notation $f_p^\wedge = f|_p^\wedge, f_p^\vee = f|_p^\vee$, etc., whenever it clarifies the presentation. Accordingly, one can decompose f as

$$\begin{aligned} f &= f_t^\wedge \wedge f_p^\wedge, \\ f &= f_t^\vee \vee f_p^\vee. \end{aligned} \tag{3.3}$$

A (•,+) operator, under which the set $\mathcal{U}$ is *stable*, can be defined computationally as an algorithm that takes UPP representations of its input functions and yields a UPP representation of the result. Considering a generic binary operator $[\cdot] * [\cdot]$, in order to compute $f * g$ we need an algorithm that computes R_{f*g} from R_f and R_g, i.e., $R_f, R_g \rightarrow R_{f*g}$. We call this *by-curve* algorithm. Such an algorithm consists of the following steps:

1. compute valid parameters T_{f*g}, d_{f*g} and c_{f*g} for the result;
2. compute intervals I_f and I_g, thus the cuts $S_f^{I_f} = \mathrm{Cut}(R_f, I_f)$ and, likewise, $S_g^{I_g}$, such that they are *sufficient* for the following step;
3. compute $S_f^{I_f}, S_g^{I_g} \rightarrow S_{f*g}^{I_{f*g}}$ where $I_{f*g} = \left[0, T_{f*g} + d_{f*g}\right[$, i.e., use an algorithm that computes the resulting sequence from cuts of the operands. We call this *by-sequence* algorithm for operator $[\cdot] * [\cdot]$;
4. return $R_{f*g} = (S_{f*g}, T_{f*g}, d_{f*g}, c_{f*g})$.

An alternative approach, for those cases where such algorithm cannot be derived (e.g., when T_{f*g} cannot be done *a priori*), is to decompose $f * g$ into an expression whose operations *can* be computed via an algorithm of this form.

The *by-curve* algorithm for operator $[\cdot] * [\cdot]$ allows us to compute the result with any operands. Unitary operators, such as lower and upper pseudoinverses, follow a similar process. Moreover, note that the distinction between *by-curve* and *by-sequence* can be easily reflected in software. In a similar fashion, the *by-sequence* algorithm may rely on a *by-element* version. This will be recalled in Chapter 8, where we use the example of the (min,+) convolution to discuss the software architecture.

Work [BT08] provides stability, UPP properties and algorithm descriptions for most DNC operators, which include minimum, addition, subtraction, (min,+) convolution and deconvolution, subadditive closure. Following similar steps, one can derive the same for maximum, (max,+) convolution and deconvolution, superadditive closure.

In the following sections, we provide UPP results for (min,+) operations discussed in [BT08]. We discuss in detail minimum and (min,+) convolution, since they both relevant for the following discussion. We also provide the equivalent result for maximum and (max,+) convolution, which can be easily derived following the same steps for the minimum and (min,+) convolution. In Chapters 10 and 11 we extend the above toolbox with lower pseudoinverse, upper pseudoinverse and composition of functions.

Remark 3.9. Parameters T_{f*g}, d_{f*g} and c_{f*g}, as well as intervals I_f and I_g, are *sufficient* to compute a representation R_{f*g} for $f * g$. There may be, in general, more than one way to compute them for an algorithm, resulting in different performance and size of the result.

Intuitively, dealing with shorter cuts leads to faster *by-sequence* algorithms. Optimized parameters and intervals can be found either by making restrictive assumptions on the shape of the operands, or by exploiting algebraic properties – we show multiple examples of this in Part II. Producing smaller representations affects, instead, the performance of its future uses. We discuss in Chapter 12 the *representation minimization* algorithm, which can be used *a posteriori* to improve the efficiency of chained operations.

3.3 Minimum of functions in $\mathcal{U}$

In this section we report the results from [BT08] about the computation of the minimum of functions in $\mathcal{U}$, $f \wedge g$. First, we exclude the case of UI functions. Indeed, it is trivial to derive that if f is ultimately $-\infty$ so is $f \wedge g$, while if f is ultimately $+\infty$ then ultimately $f(t) \wedge g(t) = g(t)$. Thus, in the following we assume both ρ_f and ρ_g to be finite.

We need to treat the following two cases separately:

- $\rho_f = \rho_g$;

- $\rho_f < \rho_g$.[4]

Proposition 3.10 (Minimum of Functions with the Same Slope). *Let f, g be functions of $\mathcal{U}$, neither of which is UI. If $\rho_f = \rho_g := \rho$, $f \wedge g$ is again a function of $\mathcal{U}$ with*

$$\begin{aligned} T &= \max\left(T_f, T_g\right), \\ d &= \operatorname{lcm}\left(d_f, d_g\right), \\ c &= \rho \cdot d. \end{aligned}$$

A proof is given in Appendix B.1.

Proposition 3.11 (Minimum of Functions with Different Slopes). *Let f, g be functions of $\mathcal{U}$, neither of which is UI. Let, without loss of generality, $\rho_f < \rho_g$, and let $\bar{t} := \frac{M_f - m_g}{\rho_g - \rho_f}$, where $M_f = \sup_{T_f \le t < T_f + d_f} \left\{f(t) - \rho_f \cdot t\right\}$ and $m_g = \inf_{T_g \le t < T_g + d_g} \left\{g(t) - \rho_g \cdot t\right\}$. Then, $f \wedge g$ is pseudo-periodic with*

$$\begin{aligned} T &= \max\left(T_f, T_g, \bar{t}\right), \\ d &= d_f, \\ c &= c_f. \end{aligned}$$

A proof is given in Appendix B.1.

The above theorems allow us to compute $T_{f\wedge g}, d_{f\wedge g}$ and $c_{f\wedge g}$ for any pair of operands f, g. Thus, it will be sufficient to compute the minimum over the interval $I_{f\wedge g} = \left[0, T_{f\wedge g} + d_{f\wedge g}\right[$. Since, for any t in $I_{f\wedge g}$, we need to compute $(f \wedge g)(t) = f(t) \wedge g(t)$, it follows that $I_f = I_g = I_{f\wedge g}$.

As for the *by-sequence* algorithm, it consists in linear comparison of sequences $S_f^{I_f}, S_g^{I_g}$, similar (in structure) to the algorithm for merging two sorted arrays. Hence, its complexity is $\mathcal{O}\left(n\left(S_f^{I_f}\right) + n\left(S_g^{I_g}\right)\right)$.

We note that, in both cases, we can find examples where these two cuts are much larger than the representations of the operands. Indeed, for the case of Proposition 3.10 we can have $\operatorname{lcm}\left(d_f, d_g\right) \gg \max\left(d_f, d_g\right)$, while for the case of Proposition 3.11 we can find curves with $\bar{t} \gg \max\left(T_f + d_f, T_g + d_g\right)$.

3.4 (min,+) convolution of functions in $\mathcal{U}$

In this section we report the results from [BT08] about the computation of the (min,+) convolution of functions in $\mathcal{U}$. These results will be instrumental for the following discussion, in two ways. First, since it is complex enough to touch many aspects

[4]Without loss of generality, as the minimum is commutative.

of the software architecture of *Nancy*, we use it to exemplify it in Chapter 8. At the same time, being a core operation in DNC, many of our algorithmic results are about improving it, thus it is worthwhile to provide a strong reference for its *base* version.

We first recall its definition, mentioning the set $\mathcal{U}$.

Definition 3.12 ((min,+) convolution)**.** Let f and g be functions in $\mathcal{U}$. We define the (min,+) convolution, for all $t \geq 0$, as

$$f \otimes g(t) := \inf_{0 \leq s \leq t} \{f(s) + g(t-s)\} \tag{3.4}$$

As mentioned in [BT08, p. 7], $f \otimes g$ is not defined if there exist t_1, t_2 such that $f(t_1) = +\infty$ and $g(t_2) = -\infty$, or vice versa (i.e., both are infinite, with opposite signs). In the following, we will assume $f \otimes g$ is *well-defined*.[5]

3.4.1 *By-curve* algorithm for (min,+) convolution

The first hurdle for the *by-curve* algorithm for (min,+) convolution is that we cannot, in general, compute *a priori* a valid period start $T_{f \otimes g}$ if $\rho_f \neq \rho_g$. However, as discussed in [BT08], the issue can be addressed if one decomposes the operands into their *transient* and (pseudo-)*periodic* parts, according to Equation (3.3). We discuss an improved result for $\rho_f = \rho_g$ later in this section. In the general case, the procedure is as follows:

1. Decompose the operands as $f = f_t^\wedge \wedge f_p^\wedge$ and $g = g_t^\wedge \wedge g_p^\wedge$.
2. Compute partial convolutions involving at least one transient part: $\otimes_{tt} := f_t^\wedge \otimes g_t^\wedge$, $\otimes_{tp} := f_t^\wedge \otimes g_p^\wedge$, $\otimes_{pt} := f_p^\wedge \otimes g_t^\wedge$.
3. Compute the partial convolution of the *periodic* parts, $\otimes_{pp} := f_p^\wedge \otimes g_p^\wedge$.
4. Finally, compute $f \otimes g = \otimes_{tt} \wedge \otimes_{tp} \wedge \otimes_{pt} \wedge \otimes_{pp}$.

Note that these steps use the property of (min,+) convolution to distribute over the minimum, i.e., $(f \wedge g) \otimes h = (f \otimes h) \wedge (g \otimes h)$. For each of these four partial convolutions, [BT08] provides the parameters T, d and c, as well as the cut intervals I_f and I_g, that allow to compute them using the *by-sequence* algorithm discussed in the following subsection. We report these results, reorganized and with some clarifications of our own, here, while proofs are provided in Appendix B.2.

Proposition 3.13 ((min,+) Convolution of *Transient* Parts)**.** *Let f and g be functions of $\mathcal{U}$. Introducing the shorthand notation $\otimes_{tt} := f_t^\wedge \otimes g_t^\wedge$, it holds that $\otimes_{tt}$ is again a function of $\mathcal{U}$, and is UI with $\otimes_{tt}(t) = +\infty$ for any $t \geq T_f + T_g$.*

[5]To be precise, [BBL18, p. 18] solves the above ambiguity, since in the (min,+) dioid $+\infty$ is *absorbing*, i.e., $(+\infty) + (-\infty) = +\infty$. However, the implementation in *Nancy* does not reflect this property, as it provides both (min,+) and (max,+) operations using a common numerical representation.

A proof is provided in Appendix B.2.

Proposition 3.14 ((min,+) Convolution of a *Transient* with a *Periodic* Part)**.** *Let f and g be functions of $\mathcal{U}$. Introducing the shorthand notation $\otimes_{tp} := f_t^\wedge \otimes g_p^\wedge$, it holds that $\otimes_{tp}$ is again a function of $\mathcal{U}$ with*

$$T_{\otimes_{tp}} = T_f + T_g, \tag{3.5}$$

$$d_{\otimes_{tp}} = d_g, \tag{3.6}$$

$$c_{\otimes_{tp}} = c_g. \tag{3.7}$$

Moreover, $\otimes_{tp}(t) = +\infty$ for any $t < T_g$.

A proof is provided in Appendix B.2. Note that, the (min,+) convolution being commutative, Proposition 3.14 applies to both $\otimes_{tp}$ and $\otimes_{pt}$.

Proposition 3.15 ((min,+) Convolution of *Periodic* Parts)**.** *Let f and g be functions of $\mathcal{U}$. Introducing the shorthand notation $\otimes_{pp} := f_p^\wedge \otimes g_p^\wedge$, it holds that $\otimes_{pp}$ is again a function of $\mathcal{U}$ with*

$$T_{\otimes_{pp}} = T_f + T_g + \operatorname{lcm}(d_f, d_g)\,, \tag{3.8}$$

$$d_{\otimes_{pp}} = \operatorname{lcm}(d_f, d_g)\,, \tag{3.9}$$

$$c_{\otimes_{pp}} = d_{\otimes_{pp}} \cdot \min\left(\frac{c_f}{d_f}, \frac{c_g}{d_g}\right). \tag{3.10}$$

A proof is provided in Appendix B.2. Next, we discuss the *cuts* of f and g necessary to compute each. For the first three terms, we can easily derive

$$\begin{aligned}
\otimes_{tt} &\to I_f = [0, T_f[\text{ and } I_g = [0, T_g[\,,\\
\otimes_{tp} &\to I_f = [0, T_f[\text{ and } I_g = [T_g, T_f + T_g + d_g[\,,\\
\otimes_{pt} &\to I_f = [T_f, T_f + T_g + d_f[\text{ and } I_g = [0, T_g[\,.
\end{aligned}$$

For the fourth term, we use the following corollary.

Proposition 3.16 (Sufficient Cuts for (min,+) Convolution of *Periodic* Parts)**.** *Let f and g be functions of $\mathcal{U}$, and let $f_p^\wedge$ and $g_p^\wedge$ be their respective* periodic *parts. Let the interval $I_{\otimes_{pp}}$ be of the form $\left[T_f + T_g, T_f + T_g + 2 \cdot d_{\otimes_{pp}}\right[$. Then, in order to compute $f_p^\wedge \otimes g_p^\wedge$ over $I_{\otimes_{pp}}$ it is sufficient to use*

$$\begin{aligned}
I_f &= \left[T_f, T_f + 2 \cdot d_{\otimes_{pp}}\right[,\\
I_g &= \left[T_g, T_g + 2 \cdot d_{\otimes_{pp}}\right[.
\end{aligned}$$

A proof is provided in Appendix B.2. From these cut intervals we can observe that, while $\otimes_{tt}$, $\otimes_{tp}$ and $\otimes_{pt}$ are not particularly complex, the convolution of periodic parts $\otimes_{pp}$ is the one subject to *hyper-period explosion*. In fact, its intervals depend on its period length $d_{\otimes_{pp}}$, which depends in turn on the least common multiple (lcm) of the period lengths of the operands, $\mathrm{lcm}(d_f, d_g)$. It follows that the number of operations required may vary considerably depending on *numerical properties* of the operands, since $\mathrm{lcm}(d_f, d_g)$ can be as small as $\max(d_f, d_g)$ or as large as the product of their numerators, $p_{d_f} \cdot p_{d_g}$.

In those cases where $d_{\otimes_{pp}} \gg \max(d_f, d_g)$, we can observe a significant impact on the computation time. Abating this phenomenon is the focus of contributions such as [GY13; LBSGY17; PLSK11] and our own discussed in Chapters 12 to 14.

As for the last step, where we compute the minimum of these four terms, we obtain the following.

Proposition 3.17 (Minimum of Decomposed (min,+) Convolution Terms). *Let f and g be functions of $\mathcal{U}$, and let $f = f_t^{\wedge} \wedge f_p^{\wedge}$ and $g = g_t^{\wedge} \wedge g_p^{\wedge}$ be their decomposition in* transient *and* periodic *parts. Let $\otimes_{tt} := f_t^{\wedge} \otimes g_t^{\wedge}$, $\otimes_{tp} := f_t^{\wedge} \otimes g_p^{\wedge}$, $\otimes_{pt} := f_p^{\wedge} \otimes g_t^{\wedge}$, and $\otimes_{pp} := f_p^{\wedge} \otimes g_p^{\wedge}$. Then*

$$f \otimes g = \otimes_{tt} \wedge \otimes_{tp} \wedge \otimes_{pt} \wedge \otimes_{pp}. \tag{3.11}$$

Moreover, $f \otimes g$ is again a function of $\mathcal{U}$ with

$$\begin{aligned} d_{f\otimes g} &= \mathrm{lcm}(d_f, d_g)\,, \\ c_{f\otimes g} &= \min\left(\frac{c_f}{d_f}, \frac{c_g}{d_g}\right) d'_{f\otimes g} = \min\left(\frac{c_f}{d_f}, \frac{c_g}{d_g}\right) \cdot \mathrm{lcm}(d_f, d_g)\,. \end{aligned}$$

A proof is provided in Appendix B.2. Algorithm 1 reports the pseudocode for the (min,+) convolution algorithm, based on the above decomposition.

(min,+) convolution of functions with the same slope

If $\rho_f = \rho_g$, we derive the following result.

Proposition 3.18 (Convolution of Curves with the Same Slope). *Let f and g be functions of $\mathcal{U}$, with $\rho := \rho_f = \rho_g$. Then $f \otimes g$ is again a function of $\mathcal{U}$ with*

$$\begin{aligned} T &= T_f + T_g + d, \\ d &= \mathrm{lcm}(d_f, d_g)\,, \\ c &= \rho \cdot d. \end{aligned}$$

The entire convolution can be computed using sequences S_f^I, S_g^I with $I = [0, T_f + T_g + 2 \cdot d[$, and then computing the convolution of these sequences $S_{f\otimes g}^I$ over the same interval.

A proof is provided in Appendix B.2.

Algorithm 1 Pseudocode for (min,+) convolution

```
    Input Functions $f$ and $g$.
    Return Their (min,+) convolution $f \otimes g$.
 1: Decompose the operands as $f = f_t^\wedge \wedge f_p^\wedge$ and $g = g_t^\wedge \wedge g_p^\wedge$
 2: Compute $h_{tt} := f_t^\wedge \otimes g_t^\wedge$, $h_{tp} := f_t^\wedge \otimes g_p^\wedge$, $h_{pt} := f_p^\wedge \otimes g_t^\wedge$ as described in [BT08]
 3: Compute $h_{pp} := f_p^\wedge \otimes g_p^\wedge$ as follows:
 4:     Compute $d = \mathrm{lcm}(d_f, d_g)$
 5:     Compute $c = d \cdot \min(\rho_f, \rho_g)$
 6:     Compute $T = T_f + T_g + d$
 7:     Compute
            $I_{f_p^\wedge} = [T_f, T_f + 2 \cdot d[$,
            $I_{g_p^\wedge} = [T_g, T_g + 2 \cdot d[$,
            $I_{\otimes_{pp}} = [T_f + T_g, T_f + T_g + 2 \cdot d[$
 8:     Compute $S_{\otimes_{pp}}^{I_{\otimes_{pp}}} = S_{f_p^\wedge}^{I_{f_p^\wedge}} \otimes S_{g_p^\wedge}^{I_{g_p^\wedge}}$
 9:     $R_{h_{pp}} = \left(S_{\otimes_{pp}}^{I_{\otimes_{pp}}}, T, d, c\right)$
10: $f \otimes g = \min(h_{tt}, h_{tp}, h_{pt}, h_{pp})$
```

3.4.2 *By-sequence* algorithm for (min,+) convolution

For the following discussion, we will consider the value of a sequence or element e, of function $f \in \mathcal{U}$ and with support I, as

$$e(t) = \begin{cases} f(t), & t \in I, \\ +\infty, & \text{otherwise.} \end{cases}$$

Thus, we can decompose a sequence S into its composing elements as

$$S(t) = e_1(t) \wedge \cdots \wedge e_n(t).$$

Let S_f be a sequence of $f \in \mathcal{U}$ with support I_f, and likewise S_g of g with I_g. Then

$$\begin{aligned} S_f \otimes S_g &= \left(e_1^f \wedge \cdots \wedge e_n^f\right) \otimes \left(e_1^g \wedge \cdots \wedge e_n^g\right) \\ &= \left(e_1^f \otimes e_1^g\right) \wedge \cdots \wedge \left(e_n^f \otimes e_n^g\right) \\ &= \bigwedge_{\substack{e_i^f \text{of} S_f, \\ e_j^g \text{of} S_g}} \left(e_i^f \otimes e_j^g\right). \end{aligned}$$

Thus, we have decomposed the *by-sequence* convolution into

- a set of *by-element* convolution,
- the *lower envelope* of the above set.

Recalling that an *element* can be either a *point* or a *segment*, there are three types of *by-element* convolution that we need to compute. These are discussed thoroughly in [BT08, Lemma 2-4], which we omit here for brevity. Suffices to say, for the following discussion, that the result of such a convolution is, in general, a sequence (of 1 or 3 elements). Thus, the set of *by-element* convolution results into a set of sequences with cardinality $\mathcal{O}\left(n(S_f) \cdot n(S_g)\right)$ – let E be the set of elements composing these sequences. The algorithm to compute the lower envelope of E, which is discussed in depth in Section 8.8, has complexity $\mathcal{O}\left(n(E) \cdot \log(n(E))\right)$.

We note that the algorithm above can be improved in various ways, both applying *algebraic* optimizations and *implementation* ones. By *algebraic* optimizations we mean that the computation is made simpler by means of applying algebraic properties to *filter* away terms. For example, as we assume (in this context) elements and sequences to have value $+\infty$ outside their support, any element that is $+\infty$ *within* its support does not affect the result, and can be omitted from the computation. Moreover, while the *by-curve* algorithm expects a sequence with a specific interval, be it I, often the cuts I_f and I_g can produce *by-element* convolutions whose support is outside I. Since these results would be discarded anyway, one can filter in advance to reduce the number of elements of which we compute the lower envelope. We mention this since, later in Part II, and in particular Section 8.7, we discuss some optimizations that are indeed implemented as algebraic filters in this step.

3.5 Other (min,+) operations

We provide here UPP results for other (min,+) operators, again from [BT08].

Proposition 3.19 (Addition). *Let f, g be functions of $\mathcal{U}$, neither of which is UI. Then $f + g$ is again a function of $\mathcal{U}$ with*

$$\begin{aligned} T &= \max\left(T_f, T_g\right), \\ d &= \operatorname{lcm}\left(d_f, d_g\right), \\ c &= \left(\rho_f + \rho_g\right) \cdot d. \end{aligned}$$

Proposition 3.20 (Subtraction). *Let f, g be functions of $\mathcal{U}$, neither of which is UI. Then $f - g$ is again a function of $\mathcal{U}$ with*

$$\begin{aligned} T &= \max\left(T_f, T_g\right), \\ d &= \operatorname{lcm}\left(d_f, d_g\right), \\ c &= \left(\rho_f - \rho_g\right) \cdot d. \end{aligned}$$

Definition 3.21 ((min,+) Deconvolution)**.** Let f and g be functions in $\mathcal{U}$. We define the (min,+) deconvolution, for all $t \geq 0$, as

$$f \oslash g(t) := \sup_{s \geq 0} \{f(t+s) - g(s)\}. \tag{3.12}$$

Proposition 3.22 ((min,+) Deconvolution)**.** *Let f, g be functions of $\mathcal{U}$, neither of which is UI. Then $f \oslash g$ is again a function of $\mathcal{U}$ with*

$$T = T_f,$$
$$d = d_f,$$
$$c = c_f.$$

3.6 Subadditive closure

Similarly to the (min,+) convolution, for the subadditive closure we do not have a closed expression for its UPP properties – rather, we have an expression whose UPP properties depend on the operand. The algorithm is based on Theorem 2.4, which we restate here for functions of $\mathcal{U}$.

Theorem 3.23 (Subadditive Closure of a Minimum)**.** *Let f and g be functions of $\mathcal{U}$. Then,*

$$\overline{f \wedge g} = \overline{f} \otimes \overline{g} \tag{3.13}$$

First, we decompose f by mapping each element of its representation into a function of $\mathcal{U}$. Let e_i, $i = 1 \dots n$, be the elements of S_f, each with support $I_i = \{t_i\}$ (if e_i is a point in t_i), or in $I_i =]t_i, t_{i+1}[$ (if e_i is a segment), and let $l+1$ be the index of the first periodic element (i.e., the point in T_f).[6] We then map each *transient* element, $i = 1 \dots l$, into a function e_i^t in $\mathcal{U}$, and each *periodic* element, $i = l+1 \dots n$, into a function e_i^p, where

$$e_i^t(t) = \begin{cases} f(t) & \text{if } t \in I_i, \\ +\infty & \text{otherwise}, \end{cases} \qquad e_i^p(t) = \begin{cases} f(t + k \cdot d) & \text{if } t \in I_i + k \cdot d, k \in \mathbb{N}_0, \\ +\infty & \text{otherwise}. \end{cases}.$$

Then, we can write a decomposition of f as $f = e_1^t \wedge \dots \wedge e_l^t \wedge e_{l+1}^p \dots \wedge e_n^p$. Then, by Theorem 3.23 we have

$$\overline{f} \overset{(3.13)}{=} \overline{e_1^t} \otimes \dots \otimes \overline{e_l^t} \otimes \overline{e_{l+1}^p} \dots \otimes \overline{e_n^p}. \tag{3.14}$$

Thus, the SAC of f is decomposed into SACs of points, open segments, periodic points and periodic open segments, for which algorithms are known [BT08]. This

[6] We assume here, for simplicity, that f *has* transient elements, thus $T_f > 0$.

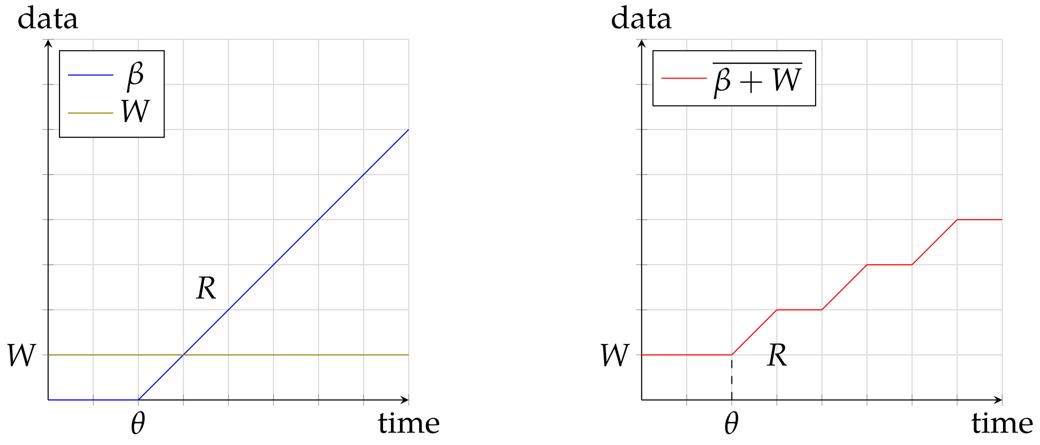


Figure 3.4: SAC $\overline{\beta + W}$ (right) when β is rate-latency and W is the ordinate of a constant function (both shown on the left).

SAC computation is $\mathcal{NP}$-hard [BT07, p. 41]: the complexity grows exponentially with the number of elements in the representation. We remark that the above algorithm makes extensive use of the convolution operation.

An exception is when β is a rate-latency curve. In this case, given a *constant function*, i.e., a function f such that $f(0) = 0$, $f(t) = W \; \forall t > 0$ where $W > 0$, SAC $\overline{\beta + f}$ can be computed in closed form [LT01, pp. 118-9]. Typically, W is a buffer dimension. We use shorthand $\overline{\beta + W}$, i.e., with the ordinate of the constant function rather than the function name itself, for better readability. As shown in Figure 3.4, the resulting function is not a rate-latency, but a staircase UPP function.

3.7 (max,+) operators

We can derive similar results for maximum, (max,+) convolution and (max,+) deconvolution, by following the same steps as the corresponding (min,+) operator with minor obvious changes – e.g., the decomposition used in the (max,+) convolution is $f = f_t^{\vee} \vee f_p^{\vee}$ instead of $f = f_t^{\wedge} \wedge f_p^{\wedge}$. We thus state them here – as they will be useful later – but omit the proofs. First, we recall the definitions of (max,+) convolution and deconvolution.

Definition 3.24 ((max,+) Convolution)**.** Let f and g be functions in $\mathcal{U}$. We define the (max,+) convolution, for all $t \geq 0$, as

$$f \overline{\otimes} g(t) := \sup_{0 \leq s \leq t} \{f(s) + g(t-s)\}. \tag{3.15}$$

Definition 3.25 ((max,+) Deconvolution)**.** Let f and g be functions in $\mathcal{U}$. We define the (max,+) deconvolution, for all $t \geq 0$, as

$$f \overline{\oslash} g(t) := \inf_{s \geq 0} \{ f(t+s) - g(s) \} . \tag{3.16}$$

As noted in [BBL18, Equation (2.13)], we can link (min,+) and (max,+) operators using the opposite.

Lemma 3.26. *Let f and g be functions of $\mathcal{U}$. Then*

$$\begin{aligned} f \otimes g &= -\left((-f) \overline{\otimes} (-g) \right), \\ f \overline{\otimes} g &= -\left((-f) \otimes (-g) \right), \\ f \oslash g &= -\left((-f) \overline{\oslash} (-g) \right), \\ f \overline{\oslash} g &= -\left((-f) \oslash (-g) \right). \end{aligned}$$

Many of the following properties can in fact be derived, other than by adapting the steps of the proof of (min,+) equivalent, by directly applying Lemma 3.26 to the (min,+) equivalent.

Proposition 3.27 (Maximum of Functions with the Same Slope)**.** *Let f, g be functions of $\mathcal{U}$, neither of which is UI. If $\rho_f = \rho_g := \rho$, $f \vee g$ is again a function of $\mathcal{U}$ with*

$$\begin{aligned} T &= \max\left(T_f, T_g\right), \\ d &= \operatorname{lcm}\left(d_f, d_g\right), \\ c &= \rho \cdot d. \end{aligned}$$

Proposition 3.28 (Maximum of Functions with Different Slopes)**.** *Let f, g be functions of $\mathcal{U}$, neither of which is UI. Let, without loss of generality, $\rho_f < \rho_g$. Let $\bar{t} := \frac{M_f - m_g}{\rho_g - \rho_f}$. Then, $f \vee g$ is pseudo-periodic with*

$$\begin{aligned} T &= \max\left(T_f, T_g, \bar{t}\right), \\ d &= d_g, \\ c &= c_g. \end{aligned}$$

Proposition 3.29 ((max,+) Deconvolution)**.** *Let f, g be functions of $\mathcal{U}$, neither of which is UI. Then $f \overline{\oslash} g$ is again a function of $\mathcal{U}$ with*

$$\begin{aligned} T &= T_f, \\ d &= d_f, \\ c &= c_f. \end{aligned}$$

For the (max,+) convolution, we provide only two results, that are particularly useful for the following discussion, i.e., the decomposition into partial (max,+) convolutions and the UPP properties of the (max,+) convolution of *periodic* parts.

Proposition 3.30 (Maximum of Decomposed (max,+) Convolution Terms)**.** *Let f and g be functions of $\mathcal{U}$. Let $f = f_t^\vee \vee f_p^\vee$ and $g = g_t^\vee \vee g_p^\vee$ be their decomposition in* transient *and* periodic *parts. Let $\overline{\otimes}_{tt} := f_t^\vee \overline{\otimes} g_t^\vee$, $\overline{\otimes}_{tp} := f_t^\vee \overline{\otimes} g_p^\vee$, $\overline{\otimes}_{pt} := f_p^\vee \overline{\otimes} g_t^\vee$, and $\overline{\otimes}_{pp} := f_p^\vee \overline{\otimes} g_p^\vee$. Then $f \overline{\otimes} g = \overline{\otimes}_{tt} \vee \overline{\otimes}_{tp} \vee \overline{\otimes}_{pt} \vee \overline{\otimes}_{pp}$. Moreover, $f \overline{\otimes} g$ is again a function of $\mathcal{U}$ with*

$$d_{f \overline{\otimes} g} = \operatorname{lcm}(d_f, d_g)\,,$$
$$c_{f \overline{\otimes} g} = \max\left(\frac{c_f}{d_f}, \frac{c_g}{d_g}\right) d'_{f \overline{\otimes} g} = \max\left(\frac{c_f}{d_f}, \frac{c_g}{d_g}\right) \cdot \operatorname{lcm}(d_f, d_g)\,.$$

Proposition 3.31 ((max,+) Convolution of *Periodic* Parts)**.** *Let f and g be functions of $\mathcal{U}$. Introducing the shorthand notation $\overline{\otimes}_{pp} := f_p^\vee \overline{\otimes} g_p^\vee$, it holds that $\overline{\otimes}_{pp}$ is again a function of $\mathcal{U}$ with*

$$T_{\overline{\otimes}_{pp}} = T_f + T_g + \operatorname{lcm}(d_f, d_g)\,, \tag{3.17}$$
$$d_{\overline{\otimes}_{pp}} = \operatorname{lcm}(d_f, d_g)\,, \tag{3.18}$$
$$c_{\overline{\otimes}_{pp}} = d_{\overline{\otimes}_{pp}} \cdot \max\left(\frac{c_f}{d_f}, \frac{c_g}{d_g}\right). \tag{3.19}$$

Proposition 3.32 (Sufficient Cuts for (max,+) Convolution of *Periodic* Parts)**.** *Let f and g be functions of $\mathcal{U}$, and let $f_p^\vee$ and $g_p^\vee$ be their respective* periodic *parts. Let the interval $I_{\overline{\otimes}_{pp}}$ be of the form $[T_f + T_g, T_f + T_g + 2 \cdot d[$. Then, in order to compute $f_p^\vee \overline{\otimes} g_p^\vee$ over $I_{\overline{\otimes}_{pp}}$ it is sufficient to use*

$$I_f = [T_f, T_f + 2 \cdot d[\,,$$
$$I_g = [T_g, T_g + 2 \cdot d[\,.$$

3.8 Plain and ultimately plain functions

In this chapter, we focused on a class of functions that is representable with a finite amount of memory, and highlighted the property of *stability*, which allows one to write algorithms for operations where both operands and results are of this same *class*. However, [BT08] discusses some pathological cases for which this stability may not be guaranteed.

Take for example, consider the subadditive closures of points or periodic points used in Section 3.6, e.g, $\overline{e_{p_a}}$ and $\overline{e_{p_b}}$, where p_a and p_b are the points $(2, 2)$ and $(3, 1)$,

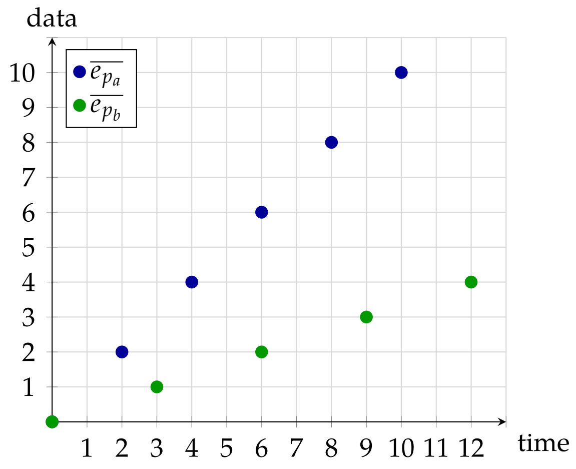


Figure 3.5: Plots of $\overline{e_{p_a}}$ and $\overline{e_{p_b}}$, which are neither plain nor ultimately plain. While $\overline{e_{p_a}} \wedge \overline{e_{p_b}}$ is not UPP, $\overline{e_{p_a}} \otimes \overline{e_{p_b}}$ is.

whose plots are shown in Figure 3.5. Consider then their minimum, $\overline{e_{p_a}} \wedge \overline{e_{p_b}}$. We can find, for any T, $t_a > T$ such that $\overline{e_{p_a}}(t_a) < +\infty$ and $\overline{e_{p_b}}(t_b) = +\infty$, as well as a $t_b > T$ such that $\overline{e_{p_a}}(t_b) = +\infty$ and $\overline{e_{p_b}}(t_b) < +\infty$. This means that neither of the two is, ultimately, always below the other, and, combined with them having different slopes, one cannot find T, d and c such that the UPP property holds. Therefore, $\overline{e_{p_a}}$ and $\overline{e_{p_b}}$ are UPP, but their minimum $\overline{e_{p_a}} \wedge \overline{e_{p_b}}$ is not.

To avoid these cases, [BT08] identifies *plain* and *ultimately plain* functions, whose definitions and properties we report below.

Definition 3.33 (Ultimately plain function)**.** A function $f \in \mathcal{U}$ is ultimately plain if $\exists T \in \mathbb{Q}_+$ such that either $\forall t > T, f(t) \in \mathbb{Q}$ or $\forall t > T, f(t) \in \{-\infty, +\infty\}$. In other words, f is, after T, either always infinite or always finite.

Definition 3.34 (Plain function)**.** A function $f \in \mathcal{U}$ is plain if either

- $f(t) \in \mathbb{Q}$ for any $t \in \mathbb{Q}_+$, or
- exist $b = \{-\infty, +\infty\}, T \in \mathbb{Q}_+$ such that
  - for any $t < T, f(t) \in \mathbb{Q}$,
  - $f(T) \in \mathbb{Q}$ or $f(T) = b$,
  - for any $t > T, f(t) = b$

Note that a function that is infinite before T but finite after T is ultimately plain, but not plain. An example of such function is $f_p^{\wedge}$, if $f(t) \in \mathbb{Q}$ for any t and $T_f > 0$. Moreover, a non-decreasing function is ultimately plain, and if $f(0) \in \mathbb{Q}$, it is also plain.

These classes then provide *sufficient* conditions for the $(\bullet,+)$ operations to be stable in $\mathcal{U}$ [BT08, Th. 2]. On the other hand, these properties are not *necessary*. For example, $\overline{e_{p_a}}$ and $\overline{e_{p_b}}$, discussed above, are neither plain nor ultimately plain, however [BT08, Prop. 9] shows that the convolution of such functions is ultimately pseudo-periodic nonetheless, hence $\overline{e_{p_a}} \otimes \overline{e_{p_b}} \in \mathcal{U}$.

As [BT08, Fig. 14] highlights, only *plain* UPP, piecewise affine, $\mathbb{Q}_+ \rightarrow \mathbb{Q} \cup \{+\infty, -\infty\}$ functions are, in general, guaranteed stability on all (min,+) operations. Note that, as summarized by same figure, plain $\mathbb{R}_+ \rightarrow \mathbb{R} \cup \{+\infty, -\infty\}$ functions may still not retain the UPP property.

In the rest of this thesis, we will consider operations that, under the hypotheses discussed in each context, provide such stability - regardless of whether or not operands are plain - hence all results are stable in $\mathcal{U}$.

Chapter 4

Used Results

In this chapter we recall the results from the literature that are used within this thesis. As mentioned in Section 3.1, the results from the literature are often based on different types of functions, and it needs to be shown if, or with what additional assumptions, they may apply to functions of $\mathcal{U}$. We will underline this, either by providing an adapted proof or by highlighting the differences, on a per-result basis.

4.1 Continuity and $(\cdot,+)$ convolution

We report here properties of (min,+) and (max,+) convolutions, which are useful for the following discussion. We report proofs, adapted for $\mathcal{U}$, in Appendix C. We start by discussing the (min,+) convolution, recalling its definition.

Definition 3.12 ((min,+) convolution)**.** Let f and g be functions in $\mathcal{U}$. We define the (min,+) convolution, for all $t \geq 0$, as

$$f \otimes g(t) := \inf_{0 \leq s \leq t} \{f(s) + g(t-s)\} \tag{3.4}$$

We highlight, for the following work, two important properties: attainability and left-continuity. These properties are proved for $\mathbb{R}_+ \to \mathbb{R} \cup \{+\infty, -\infty\}$ functions in, respectively, [BBL18, Proposition 3.10] and [BBL18, Proposition 3.11]. We adapt these properties for functions of $\mathcal{U}$, and provide the proofs in Appendix C.[1]

Proposition 4.1 (Attainability of (min,+) Convolution)**.** *Let f and g be left-continuous, non-decreasing functions of $\mathcal{U}$. Then, for any $t \in \mathbb{Q}_+$ it exists $s^* \in [0,t]$ such that*

$$f \otimes g(t) = \inf_{0 \leq s \leq t} \{f(s) + g(t-s)\} = f(s^*) + g(t-s^*).$$

In other words, the infimum is always attained in $[0,t]$.

[1]Note that the proofs in [BBL18] use the completeness property of $\mathbb{R}$, which does not apply to $\mathbb{Q}$. We show in Appendix C that the resulting $t^* \in \mathbb{R}$ must be, by construction of $f \in \mathcal{U}$, also in $\mathbb{Q}$.

A proof is provided in Appendix C.2.

Proposition 4.2 (Left-Continuity of (min,+) Convolution). *Let f and g be left-continuous, non-decreasing functions of $\mathcal{U}$. Then, $f \otimes g$ is left-continuous.*

A proof is provided in Appendix C.2.

Analogous properties can be derived for the (max,+) convolution.

Definition 3.24 ((max,+) Convolution). Let f and g be functions in $\mathcal{U}$. We define the (max,+) convolution, for all $t \geq 0$, as

$$f \overline{\otimes} g(t) := \sup_{0 \leq s \leq t} \{f(s) + g(t-s)\}. \tag{3.15}$$

Proposition 4.3 (Attainability of (max,+) Convolution). *Let f and g be right-continuous, non-decreasing functions of $\mathcal{U}$. Then, for any $t \in \mathbb{Q}_+$ it exists $s^* \in [0,t]$ such that*

$$f \overline{\otimes} g(t) = \sup_{0 \leq s \leq t} \{f(s) + g(t-s)\} = f(s^*) + g(t-s^*).$$

In other words, the supremum is always attained in $[0,t]$.

A proof is provided in Appendix C.3.

Proposition 4.4 (Right-Continuity of (max,+) Convolution). *Let f and g be right-continuous, non-decreasing functions of $\mathcal{U}$. Then, $f \overline{\otimes} g$ is right-continuous.*

A proof is provided in Appendix C.3.

4.2 Subadditive functions

We recall here the definition and properties of subadditive functions, highlighting that they still apply in $\mathcal{U}$. Proofs can be found in [LT01], which generally assumes non-decreasing functions. The alert reader can verify that said proof hold also without this property, hence for functions of $\mathcal{U}$.

Definition 4.5 (Subadditive Function). Let f be a function of $\mathcal{U}$. We say f is subadditive if and only if $f(u) + f(s) \geq f(u+s)\ \forall u, s \geq 0$.

The following, which we already mentioned, is reported as Theorem 3.1.11 [LT01].

Theorem 3.23 (Subadditive Closure of a Minimum). *Let f and g be functions of $\mathcal{U}$. Then,*

$$\overline{f \wedge g} = \overline{f} \otimes \overline{g} \tag{3.13}$$

Theorem 4.6 (Convolution of Subadditive Functions). *Let f and g be subadditive functions of $\mathcal{U}$. Then, so is $f \otimes g$.*

The following is reported as Corollary 3.1.1 in [LT01].

Corollary 4.7. *Lef f be a subadditive function of $\mathcal{U}$ such that $f(0) = 0$. Then*

$$f \otimes f = f, \tag{4.1}$$
$$\overline{f} = f. \tag{4.2}$$

A related property that, although not limited for subadditive functions, we will use in that context, is reported as Rule 8 in [LT01, p. 113].

Lemma 4.8. *Let f and g be functions of $\mathcal{U}$ such that $f(0) = g(0) = 0$. Then*

$$f \wedge g \geq f \otimes g \tag{4.3}$$

4.3 Upper and lower pseudoinverses

Lower and upper pseudoinverses are operators useful in many contexts. For example, work [Lie17] shows that (min,+) and (max,+) algebra can be interpreted as mirror images of each other, as one can use these operators to switch between the two. Furthermore, they are useful for implementation of algorithms such as the one for composition that we describe in Chapter 11, and, together with the composion, to compute the horizontal deviation between any two curves as shown by Equation (2.3).

Definition 4.9 (Lower and Upper Pseudoinverse). Let $f \in \mathcal{U}$ be non-decreasing. Then its *lower pseudoinverse* is defined as

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

and its *upper pseudoinverse* is defined as

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

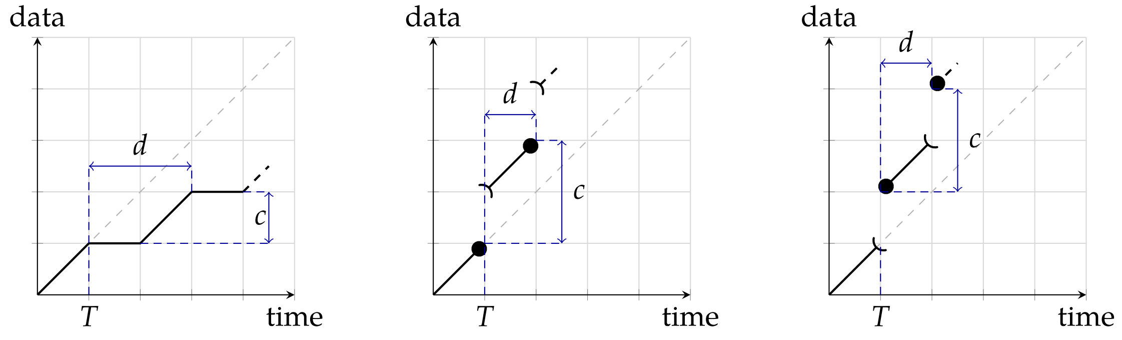
We can find an equivalent definition as follows.

Proposition 4.10. *Let $f \in \mathcal{U}$ be non-decreasing. For all $y > f(0)$, its lower pseudoinverse is equal to*

$$f_{\downarrow}^{-1}(y) = \sup \left\{ t \geq 0 \mid f(t) < y \right\}, \tag{4.4}$$

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

$$f_{\uparrow}^{-1}(y) = \inf \left\{ t \geq 0 \mid f(t) > y \right\}. \tag{4.5}$$



(a) f. (b) $f_{\downarrow}^{-1}$. (c) $f_{\uparrow}^{-1}$.

Figure 4.1: Example of lower and upper pseudoinverse of a function f.

Note that [Lie17] reports a slightly different definition, because functions are defined in $\mathbb{R} \to \mathbb{R} \cup \{+\infty, -\infty\}$. Our functions in $\mathcal{U}$ are defined in $\mathbb{Q}_+ \to \mathbb{Q} \cup \{+\infty, -\infty\}$, hence our domain is lower bounded. The consequences of this difference are discussed in Appendix D.1, which also contains a proof of Proposition 4.10.

An example of these operators is provided in Figure 4.1, which shows a function of $\mathcal{U}$ and its lower and upper pseudoinverses.

As shown in [Lie17, p. 64, Lemma 10.1], the lower pseudoinverse is always left-continuous and the upper pseudoinverse is right-continuous. We restate this result below, for functions in $\mathcal{U}$.

Lemma 4.11. *Let $f \in \mathcal{U}$ be non-decreasing. Then, $f_{\downarrow}^{-1}$ is left-continuous and $f_{\uparrow}^{-1}$ is right-continuous.*

A proof is provided in Appendix D.2. The above is consistent with the typical usage of $(\bullet,+)$ algebra: in $(\min,+)$ functions are assumed to be left-continuous, whereas in $(\max,+)$ they are assumed to be right-continuous.

Moreover, as shown in [Lie17, Lemmas 10.1c, 10.1d], upper and lower pseudoinverses can be combined to yield something close to involutive properties (again, proofs for $\mathcal{U}$ are provided in Appendix D.2).

Lemma 4.12. *Let $f \in \mathcal{U}$ be non-decreasing and left-continuous. Then,*

$$f = \left(f_{\uparrow}^{-1}\right)_{\downarrow}^{-1}. \tag{4.6}$$

Lemma 4.13. *Let $f \in \mathcal{U}$ be non-decreasing and right-continuous. Then,*

$$f = \left(f_{\downarrow}^{-1}\right)_{\uparrow}^{-1}. \tag{4.7}$$

In [Lie17, p. 61], it is reported for any non-decreasing $f : \mathbb{R} \to \mathbb{R} \cup \{+\infty, -\infty\}$ that $f_{\downarrow}^{-1} \leq f_{\uparrow}^{-1}$. For the functions discussed therein, it is an obvious property as the equivalences in Equations (4.4) and (4.5) always hold. For functions of $\mathcal{U}$, it is less obvious as the claim is only true for interior points.

Lemma 4.14. *Let f be a non-decreasing function of $\mathcal{U}$. Then, for all $y \geq f(0)$,*

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

We provide a proof in Appendix D.1.

Lastly, in [Lie17, p. 62] it is shown the following.

Proposition 4.15. *Let f be a non-decreasing function of $\mathcal{U}$. Then, for all $x, y \in \mathbb{Q}_+$*

$$f(x) > y \implies f_{\uparrow}^{-1}(y) \leq x, \tag{4.8}$$

$$f(x) \leq y \implies f_{\uparrow}^{-1}(y) \geq x, \tag{4.9}$$

$$f(x) < y \implies f_{\downarrow}^{-1}(y) \geq x, \tag{4.10}$$

$$f(x) \geq y \implies f_{\downarrow}^{-1}(y) \leq x. \tag{4.11}$$

4.4 Isomorphism between (min,+) and (max,+) algebra

Works [PLSK11; Lie17] show that operations in one algebra can be mapped to the other via *pseudoinversion* of operands and results. This *isomorphism* has also been observed to provide, in some cases, an improvement in computation runtime [PLSK11]. We thus report here the results from [Lie17], adapted to $\mathcal{U}$, which we will use in Chapter 14 to provide a novel, improved, algorithm for (•,+) convolutions. The proofs of this section are provided in Appendix F.1. The first result, adapted from [PLSK11, Theorem 1] and [Lie17, Theorem 10.3b], shows the link between (min,+) and (max,+) convolution, for left-continuous functions, via upper pseudoinverse.

Theorem 4.16 (Isomorphism of Convolution For Left-Continuous Functions)**.** *Let f and g be functions of $\mathcal{U}$ that are left-continuous and non-decreasing. Then,*

$$(f \otimes g)_{\uparrow}^{-1} = \left(f_{\uparrow}^{-1}\right) \overline{\otimes} \left(g_{\uparrow}^{-1}\right). \tag{4.12}$$

We provide a proof in Appendix F.1.

As a consequence, we can apply Proposition 4.2 and Lemma 4.12 and obtain the following.

Corollary 4.17 (Alternative Expression for (min,+) Convolution, via its Isomorphism)**.** *Let f and g be functions of $\mathcal{U}$ that are left-continuous and non-decreasing. Then,*

$$f \otimes g \overset{(4.6)}{=} \left((f \otimes g)_{\uparrow}^{-1}\right)_{\downarrow}^{-1} \overset{(4.12)}{=} \left(f_{\uparrow}^{-1} \overline{\otimes} g_{\uparrow}^{-1}\right)_{\downarrow}^{-1}. \tag{4.13}$$

The analogous result for right-continuous functions, using the lower pseudoinverse, is provided in [Lie17, Theorem 10.4b].

Theorem 4.18 (Isomorphism of Convolution For Right-Continuous Functions)**.** *Let f and g be functions of $\mathcal{U}$ that are right-continuous and non-decreasing. Then,*

$$(f \overline{\otimes} g)_{\downarrow}^{-1} = \left(f_{\downarrow}^{-1}\right) \otimes \left(g_{\downarrow}^{-1}\right). \tag{4.14}$$

We provide a proof in Appendix F.1.

We can apply Proposition 4.4 and Lemma 4.13 and obtain the following.

Corollary 4.19 (Alternative Expression for (max,+) Convolution, via its Isomorphism)**.** *Let f and g be functions of $\mathcal{U}$ that are right-continuous and non-decreasing. Then,*

$$f \overline{\otimes} g \overset{(4.7)}{=} \left((f \overline{\otimes} g)_{\downarrow}^{-1}\right)_{\uparrow}^{-1} \overset{(4.14)}{=} \left(f_{\downarrow}^{-1} \otimes g_{\downarrow}^{-1}\right)_{\uparrow}^{-1}. \tag{4.15}$$

$$\tag{4.16}$$

We note that the assumptions of these properties are not restrictive: in practice, it is in fact the norm for curves in (min,+) algebra to be non-decreasing and left-continuous, while curves in (max,+) algebra are usually non-decreasing and right-continuous.

Chapter 5

Related Works

The theory of Deterministic Network Calculus dates back to the early 1990s, and it is mainly due to the work of Cruz [Cru91a; Cru91b], Le Boudec and Thiran [LT01], and Chang [Cha00]. DNC focuses on finding *deterministic* upper bounds to the worst-case performance of systems, e.g., an upper bound to the worst-case delay, using (min,+) and (max,+) algebra [BCOQ92]. Its *stochastic* counterpart, Stochastic Network Calculus (SNC), was soon introduced by works [Cha00; CBL06; Fid06; JL08; CS12], to take advantage of statistical multiplexing and obtain more efficient dimensioning for use cases where a few violations of the bound are tolerable. In both, NC provides scheduling abstractions and composable analysis, allowing to reduce the study of a large network in a more manageable (but not necessarily not computationally costly) problem.

Since then, a considerable number of papers have extended the DNC framework to include different scheduling algorithms and flow multiplexing schemes, as well as network architectures, topologies, and applications. Such works cover FIFO multiplexing [LMMS06; BLMS10; BLMS12; BS12; BS15], round-robin multiplexing schemes [BSS12a; BBL18; Bou21; TLB21; TB21; TL22] as well as arbitrary multiplexing [SZF08; BJT10; BT16]. Deterministic Network Calculus has been applied to sensor networks [SR05; SZT07; ZYD11; SBP17], avionic network [CSEF06; BSF09; BSF10; FE17; SLSF18], networks-on-chip [BSGE09; QLD09; QLD10; GM18; GM19; BGDM20], manufacturing systems [BJLL06; KGNMP18] as well as Time-Sensitive Networking (TSN) [ZCWW19; MSB19; MHG20; ZPZDB21; MSB22].

The *computational* aspects of DNC implementations have been the subject of several papers in the past. Problems such as efficient data structures to represent arrival/service curves or functions, or the complexity of DNC operators (e.g., convolution or subadditive closure) have been tackled in the work of Bouillard and Thierry [BT08] and the related technical report [BT07], which discusses the implementation aspects in more detail. These findings find a thorough exposition in book [BBL18], which also reviews the implementations of several existing tools. The idea of piecewise affine, $\mathbb{Q}_+ \to \mathbb{Q} \cup \{+\infty, -\infty\}$, UPP curves as a class closed with respect to DNC operations is in fact reported in these works, and we use the DNC algorithms described therein as a baseline.

A related research field is that of Real-Time Calculus (RTC), developed for real-time systems [TCN00; Wan06]. RTC uses (min,+) and (max,+) algebra to obtain the output of a system given its input, similarly to DNC. For this reason, the two methodologies have often evolved via cross-fertilization, with solutions devised in one context often being ported to the other. Moreover, a formal link between the two is provided in [BJT09]. Indeed, we believe that *Nancy*, as it supports both (min,+) and (max,+) operations, may be useful also for RTC researchers. Similarly, [RQB22] shows how Response Time Analysis, another formalism used for verification of real-time properties, can be linked to DNC.

To the best of our knowledge, there are no public, open-source *libraries* that implement DNC operations working on UPP curves. By "library", we mean a set of reusable implementations of (•,+) algebra operations, providing ease of constructing and manipulating curves, and allowing a user to extend and specialize its algorithms for improved efficiency (e.g, by adding new code paths or subclasses) or extending its use cases. The *COINC* library [BCGHLL09] did support operations on UPP curves, implementing, as we do, the results of [BT08], but it appears that it is no longer available. The two existing libraries that can handle UPP curves are the *RTC Toolbox* [WTa; Wan06] and the *RTaW-Pegase* library [RTaWc].

The *RTC Toolbox* library is a publicly available Java library, often used via its MATLAB integration. While publicly and freely available, its source code is not, which limits the ability to verify or debug it, or to extend it to cover lack of features. We discuss its features and provide a comparison, albeit superficial, with *Nancy* in Chapter 15.

The *RTaW-Pegase* library is proprietary commercial product. An online interpreter is freely available to try its functionalities [RTaWb]. It is *claimed* that the library implements efficiently a large set of operations for UPP curves. However, its license prohibits to disclose both benchmarking and test results, hence we shall not provide any comment on such claims or comparison with *Nancy*.[1]

There are, instead, several DNC *network analysis tools* available, which are software packages that implement methods for analyzing specific types of networks. These often implement their own versions of (•,+) algebra operations, which are typically restricted to classes of curves that match the type of network they are designed to analyze. Many are mentioned in [BBL18], while a review of their capabilities is reported in [ZHLC20]. For example, *DEBORAH* [BLMS10] analyzes FIFO tandems using a linear programming approach, which limits the supported inputs

[1]*Customer [...] may not disclose any information regarding any benchmark or tests of the Product to any third party.* [RTaWa, End-user License Agreement]

to rate-latency service curves traversed by leaky-bucket-shaped flows. Therefore, it only implements (min,+) convolution and subtraction of pseudo-affine curves, which are algorithmically trivial. *NC-TANDEM-TIGHT* [BJT10; BT16] analyzes arbitrary multiplexing tandems by modeling their worst-case delay computation as a linear programming problem, which implicitly rules out generic UPP curves, which would yield non-convex programs instead. Its parallel work on FIFO multiplexing [BS12; BS15], which improves on the results of *DEBORAH*, suffers from the same limitation. *NC-Maude* [Boy10] is a tool written in Maude, a high-performance reflective language. It uses a rational numeric type instead of floating point, however it supports only simple functions such as rate-latency and token-bucket as inputs. It was released as an open and extensible tool, although it appears to be no longer available. The *DiscoDNC* tool [BS14], now *NCorg DNC*, is limited to ultimately concave/convex piecewise affine curves, for which DNC operations are considerably simpler. It supports using *RTC Toolbox* as its underlying computational library [SB17], however the latter is used under the same constraints of ultimately concavity/convexity, without exploiting the broader modelling capabilities. Work [LHL17b] describes a tool written in the NVIDIA CUDA language that computes convolutions and deconvolutions using GPUs, claiming improved efficiency over standard CPU-based computation. However, there seems to be no executable or code to go with this paper.

Unfortunately, none of these tools, or the other mentioned in [BBL18; ZHLC20] provide reusable, general-purpose DNC operations.

From the point of view of algorithmic performance, several works discuss on techniques to improve the runtime of DNC and RTC studies. RTC work [GY13] first observed that multi-hop traversal – which entails chained convolutions – is subject to state explosion, and that the latter makes convolutions exponentially complex. They proposed a way to mitigate this problem, which relies on inferring the maximum time beyond which the shape of the resulting functions is immaterial, which turns out to be considerably smaller than the lcm of the periods, thus leading to more efficient operations. This idea of a compact domain is transferred to DNC in [LBS16] – allowing it to be used in conjunction with DNC service curves. Work [LBSGY17] further generalizes it to more operations and more general RTC settings. DNC analysis limited to compact domains [LBS16] consists in finding finite upper bounds to the time where operations should be computed. This allows *by-sequence* operations to be computed between two finite sets of elements – which one can imagine as *transient* parts – disregarding periodicity and the lcm explosion that comes with it. In these techniques, the upper bound is chosen so that the end-to-end delay and backlog analysis is not affected. Such a bound can be found by working with lower and upper approximations of *both* the arrival curve and the service curves, using CPL curves, which is computationally inexpensive. However, this method requires *superadditivity* of service curves (Definition 2.6), a property that does not hold in many settings, e.g., in tandems networks of flow-controlled links, where curves are *subadditive* instead (Definition 2.2).

Another approach, introduced in [LCH14], uses containers and inclusion functions. This approach provides, for the result of a computation, an upper convex bound and a lower concave bound, trading off the accuracy of computing the *actual* result with the efficiency of computing such bounds, as the algorithm complexity becomes linear for the convolution and quasi-linear for the subadditive closure.

Systems with flow control have traditionally been analyzed using Markov Chains [Bal11], under the name of "queueing systems with blocking". That method allows one to find mean performance indexes (and, possibly, distributions), starting from a *stochastic* characterization of input traffic and service. The first works analyzing flow control in the framework of DNC have been [Cha97], [ACOR99]. The exact method – i.e., the one using nested SACs – is a direct application of these results. The approximate method – i.e., the one using convolution of SACs – is instead shown in [BJLL06]. This paper, however, does not assess the gain in efficiency warranted by the approximate method, nor it acknowledges the fact that it seems to preserve accuracy. We argue that this may be due to the fact that the computational problems addressed in this thesis were in the way of such an evaluation. A different use case with hop-by-hop flow control is studied in [BPC09], which focuses on Stream Processing Systems in Real-Time Calculus. It is shown therein [BPC09, Theorem 3] that an effective service curve for the first node in a tandem can be computed via a chain of convolutions of subadditive expressions, i.e., the same type whose computation we optimize in this thesis. Paper [PS20] uses a DNC model with flow control to model Denial-of-Service (DOS) attacks. Flow control is also addressed in [BS17], in the framework of SNC.

Work [PLSK11] was first to observe the isomorphism between (min,+) and (max,+) convolution and its utility in reducing computation time. They observed that, to compute (min,+) convolutions of *event-based* service curves, replacing this with (max,+) convolutions using pseudoinverses was considerably faster. They were, however, unable to discern whether the improvement was due to algebraic properties or inefficiencies in the implementation of *RTC Toolbox*. In Chapter 14, we discuss the algebraic properties that justify their observations, and expand them with novel results.

Chapter 6

Research Statement

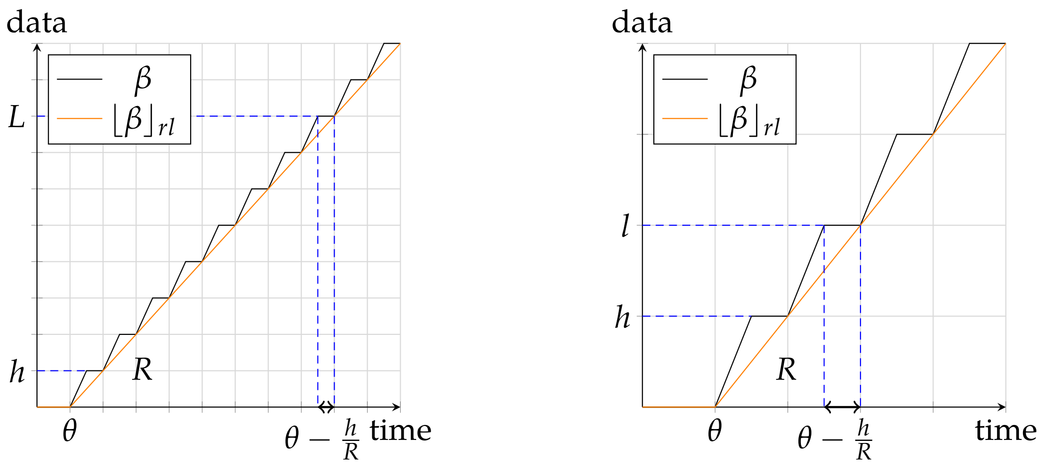
In the previous chapters, we have shown some DNC results, the mathematical model we use and associated useful results. We can now state the objectives of this thesis with more clarity than in Chapter 1.

In Deterministic Network Calculus, it is often the case that studies involve systems and processes that can be modelled with simple non-decreasing, non-negative, convex or concave models – such as the rate-latency service curve and the token-bucket arrival curve. However, it is also often the case that the service curve describing a node, such as a round-robin scheduler or a flow-controlled link, is not a simple convex curve, or not even a Ultimately Affine one. Thus, complex UPP curves naturally occur in nontrivial systems. Performing operations on these is *complex* from a computational standpoint: state explosion (see Chapter 3) tend to occur, due to the lcm operation required to compute the hyperperiod.

Of course, one can envisage a simple, but crude approach that computes *approximated* end-to-end service curves that lower bound each resulting UPP curve with a rate-latency curve. This would certainly make computations considerably faster, but may entail a considerable loss of accuracy. We exemplify this using a simple UPP curve $\beta = \beta_{R,\theta} \otimes \overline{\beta_{R,\theta} + h}$, with $R \cdot \theta > h$, i.e., the equivalent service curve of a flow-controlled link. In this case, such lower bound $\lfloor \beta \rfloor_{rl}$ would have $\theta_{lb} = \theta, R_{lb} = \frac{h}{\theta}$, and the error introduced by it is upper bounded by $\theta - \frac{h}{R}$.

As shown in Figure 6.1, the impact of such error on the end-to-end delay depends on the characteristic of the input traffic. Notably, small messages would incur relatively larger penalty than large messages, and the loss in accuracy would be non-negligible. A similar result is obtained using the container approach discussed in [LCH14], which is based on convex and concave bounds.

In other cases, though, this complexity is part of the result itself, and cannot be simply avoided with such a lower bound. Consider for example the results of [TLB21], in particular the per-flow strict service curve of an IWRR scheduler that we reported in Theorem 2.7. In this case, the service of the scheduler is computed



(a) Delay bound for a long message of length L. The one obtained using $\lfloor\beta\rfloor_{rl}$ is overestimated by $\theta - \frac{h}{R}$, 6% w.r.t. the one obtained using β.

(b) Delay bound for a short message of length l. The one obtained using $\lfloor\beta\rfloor_{rl}$ is overestimated by $\theta - \frac{h}{R}$, 20% w.r.t. the one obtained using β.

Figure 6.1: Delay overestimation introduced by lower-bounding a staircase UPP curve with a rate-latency curve.

through multiple steps that involve UPP curves.

However, the software support publicly available is quite lacking, with few options to researchers to perform or reproduce such studies, and even fewer if one seeks to extend such software towards specific research interests. Thus, we sought out to implement the popular UPP algorithmic toolbox, described in [BT07; BT08; BBL18], into an extensible open source library, *Nancy*. Such library is meant to be accessible and verifiable by the research community.

However, even *with* the software at hand, some studies still appear to be unfeasible due to their computational cost. This calls for more research in the algorithmic aspects of DNC. On one hand, this involves improving the implementation so that the capabilities of the hardware is fully realized, e.g., by identifying parallelizable parts of the code and avoiding performance loss due to saturation of available memory. On the other hand, this involves research into the algebraic aspects, so that properties of the operands – other than the simple fact of being UPP – can be used to improve the runtime of our computations.

Moreover, some operators, like pseudoinverses and composition, were not addressed within the UPP algorithmic toolbox, which leaves important gaps towards implementation of many recent studies that make of them – like the [TLB21] result that we just recalled. Thus, we sought out the additional goal of extending the UPP algorithm toolbox both to improve the algorithmic properties of some operations of practical interest, like chains of (min,+) convolutions, and to provide support – backed by formal results – to the mentioned operators. This is done leveraging the aforementioned extensibility of *Nancy*.

In the following sections, we restate the use cases that we will use as motivating examples to address, and how we plan to improve upon them in Part II.

6.1 Efficient subadditive convolutions: flow-controlled tandem

Consider a flow traversing a tandem network of n flow-controlled nodes. Each node i offers to that flow a service curve β_i. After node i, $i < n$, there is a flow control window W_{i+1}. We initially assume that the flow control is instantaneous. Our goal is to compute an end-to-end service curve of the above tandem network. This will allow one to compute a bound on the end-to-end delay and backlog, if the flow itself has an arrival curve. We want to be able to do this efficiently.

We first show that computing an end-to-end service curve, whether the exact (Equation (2.8)) or the approximate one (Equation (2.10)), incurs state explosion and may require very long computation times, even when n is small (e.g., three nodes).

The experimental setup we use to show this uses a four-hop tandem of flow-controlled nodes. We assume that nodes have rate-latency service curves, $\beta_i = \beta_{R_i,\theta_i}$, and their parameters are those in Table 6.1.

We chose to use two different sets of parameter values to better highlight the issues – and, later, the impact of the optimizations. As the length and/or feasibility of the computations depend on the parameters chosen, we found that a setting that can be solved with the exact method is often trivial with the approximate one, and a setting that is hard with the approximate method is often computationally intractable with the exact one. Therefore, a single set of parameters would not suffice to clearly demonstrate both.

We attempt to compute the end-to-end equivalent service curves, via the exact and approximate methods, using the algorithms described in [BT08], on a desktop PC (System 1 in Appendix H). We report computational results in, respectively, Table 6.2 and Table 6.3, where we highlight both the representation size of the results and the time it takes to compute them. The alert reader will notice that the results reported in the tables are intermediate results towards the equivalent end-to-end service curves via (2.8) and (2.10), respectively. We cap computation times at 24 hours.

The above results show that computation times are significant, and that state explosion does occur, even with the approximate method. As mentioned in Chapter 5, we cannot abate these computation times by working on *compact domains*, as

Table 6.1: Parameters of the example tandem network.

	Exact			Approximate		
i	β_i rate	β_i latency	W_{i+1}	β_i rate	β_i latency	W_{i+1}
1	8	5	3	21	15	23
2	11	7	7	30	17	29
3	12	4	3	7	27	20
4	1	5		21	20	

Table 6.2: Computational results, exact method.

comp. time of $\overline{\beta_2 \otimes \beta_3^{eq} + W_3}$	6 h 24 m
$n(\beta_2 \otimes \beta_3^{eq} + W_3) \to n\left(\overline{\beta_2 \otimes \beta_3^{eq} + W_3}\right)$	$10 \to 10600$
comp. time of $\overline{\beta_1 \otimes \beta_2^{eq} + W_2}$	> 24 h
$n(\beta_1 \otimes \beta_2^{eq} + W_2) \to n\left(\overline{\beta_1 \otimes \beta_2^{eq} + W_2}\right)$	*unknown*

Table 6.3: Computational results, approximate method.

comp. time of $\beta_{\{1,3\}} = \overline{\beta_1 \otimes \beta_2 + W_2} \otimes \overline{\beta_2 \otimes \beta_3 + W_3}$	0.14 s
$n\left(\overline{\beta_1 \otimes \beta_2 + W_2}\right), n\left(\overline{\beta_2 \otimes \beta_3 + W_3}\right) \to n\left(\beta_{\{1,3\}}\right)$	$6, 6 \to 270$
comp. time of $\beta_{\{1,4\}} = \beta_{\{1,3\}} \otimes \overline{\beta_3 \otimes \beta_4 + W_4}$	6 h 13 m
$n\left(\beta_{\{1,3\}}\right), n\left(\overline{\beta_3 \otimes \beta_4 + W_4}\right) \to n\left(\beta_{\{1,4\}}\right)$	$270, 6 \to 1456$

suggested in [LBS16]. In fact, that method requires that the service curves involved are *superadditive*. In our model, the operands of these tough convolutions are instead *subadditive*, because they are the result of SACs. We are not aware of any method that allows one to limit the domains in this case.

Our approach to gaining efficiency is to abate both the number and the computation runtime of the convolutions of subadditive UPP curves. This operation, in fact, lies at the core of both the exact and the approximate methods (recall that the SAC of a UPP curve can be computed as a convolution the SACs of its elements, as explained in Section 3.6). Reducing their number and making them as fast as possible is therefore going to make both methods more efficient. We do this *without introducing approximations*: our computations are always exact. In the contribution of this thesis, we show how we accomplish this, leveraging both *representation minimization* and algebraic tools: first, we show that minimizing the representation R_f of the functions f involved in the operations may provide remarkable benefits. Then, we present three theorems that can be used to reduce the computation time of convolutions, leveraging subadditivity of the operands.

We spend a few words discussing the generality of this example. With flow-controlled networks, different models can be envisaged as far as:

1. the exact place where flow control stops traffic when the flow control window is closed, w.r.t. to the service curve modeling the sending node. For example, this may be an input buffer of the sending node, or an output buffer instead. The alert reader can check that using one or the other will lead to slightly different expressions for both the exact and the approximate end-to-end service curves. However, they will still be of the same type as (2.8) and (2.10), respectively, i.e., with either nested SACs or convolutions of subadditive UPPs. Therefore, any computational issue that we address throughout this thesis will still be present;

2. whether or not the return path, i.e., the flow control credit feedback, is instantaneous. Depending on how such feedback is implemented, other models may, for instance, include a service curve on the return path as well. Again, this does not change the structure of the expressions that we seek to compute efficiently.

6.2 Extending the algorithmic toolbox: Interleaved Weighted Round Robin

In Section 2.4, we recalled various results from the literature that make use of pseudoinverses and composition, operators whose UPP properties are not discussed in the work of Bouillard et al. [BT07; BT08; BBL18]. This includes an expression for computing the horizontal deviation (Equation (2.3)) and many models for round-robin schedulers, among which we highlighted [TLB21, Theorem 1]. We selected this result as it provides a multistep process that applies various operations on UPP curves, including composition which, as we will show, in turn requires pseudoinverses. It is thus a proper use case to exemplify our software support for DNC results, and show its capabilities also by comparing the expressiveness of the code to that of the mathematical result.

Lastly, the curves produced by such model, being non subadditive UPP functions, may still lead, based on numerical properties, to computationally taxing convolutions. So, there is still a need for further optimizations that address a more generic case than the one discussed in previous section.

6.3 Improving convolution runtime using isomorphism

Moreover, we also researched improvement of convolutions that were not addressed by the work on flow-controlled tandems, i.e., non-subadditive UPP curves. As mentioned, the work [PLSK11] was the first to observe that the isomorphism between (min,+) and (max,+) convolution (Theorem 4.16) can be exploited to reduce the runtime of (min,+) convolutions, in particular when chained. They were, however, unable to discern whether the improvement was due to algebraic properties or inefficiencies in the implementation of *RTC Toolbox*.

Thus, we set out to investigate and identify, exploiting the insights gained on pseudoinverses of UPP functions, the algebraic roots of this improvement. We found out that, on one hand, the improvement was algebraically explainable – and thus not a quirk of the *RTC Toolbox* – but on the other hand, that the improvement was not *guaranteed*, and counterexamples can be built that suffer worse runtime if the isomorphism is applied. We then produce a novel technique, which exploits properties of both the "(min,+) computation" and the "(max,+) computation" to obtain an algorithm that is, in most cases, better than both, and is at worse as good as the best of the two (plus a negligible overhead).

Note that these results hold regardless of the shape of the curves – whether subadditive or not. Compared to the optimizations described in Section 6.1, though, these still require to perform a convolution (even if with improved parameters) while the improvements on subadditive convolutions largely avoid them, replacing them instead with minima. Thus, in practice, these optimizations do not overlap.

Part II

Contribution

Chapter 7

Overview

In the second part of this thesis, we outline our contribution. As mentioned in the introduction, this contribution has two natures, as it is, on one hand, a product of software engineering, addressing the challenges that arise in the implementation of an efficient and extendable computational library, and, on the other hand, it provides novel algebraic results, which extend the mathematical framework leading to significant algorithmic optimizations. We try, then, to put these aspects side by side, to show the direct link between formal results and an efficient implementation in software.

In Chapter 8, we introduce the core design choices and architecture of the *Nancy* library and its project. We discuss a few core algorithms (from `Cut()` to (min,+) `C⌋ onvolution()`) and show, using simplified code (but not *pseudo*code), the process of optimizing them with respect to memory utilization and parallelization. We then mention how *Nancy* is made publicly available, its testing process, its documentation, its support for notebooks – aspects that are, if hardly significant from a purely theoretical point of view, of fundamental nature for a software project that aims to satisfy its design goals.

In Chapter 9, we discuss some features of *Nancy* that, while outside the core set of operations described by the UPP algorithmic toolbox, are highly useful to a user. The use cases considered include the implementation of new operations, verification of properties and/or hypothesis, writing tests, advance new research. These are relevant examples of the larger *rich API* we refer to.

In Chapters 10 and 11, we extend the algorithmic toolbox presented in [BT08] with, respectively, pseudoinverses and composition. We do so by providing, on one hand, the formal results for stability of the set of function $\mathcal{U}$ over these operators and their UPP properties, and on the other, the algorithmic steps to compute them.

In Chapter 12, we discuss the *representation minimization*, outlining the issue of non-minimal representations and their effects on computation time, and providing an algorithm to reduce any representation to their minimal equivalent.

In Chapter 13, we discuss the (min,+) convolution of subadditive curves, and provide novel formal results that allows us to abate its computational cost significantly – in some cases, avoiding the convolution entirely. We show that in the use case of flow control tandems, presented in Section 6.1, these optimizations reduce the computation times by several orders of magnitude.

In Chapter 14, we discuss the use of the isomorphism between (min,+) and (max,+) convolution to improve both algorithms. We discuss the results of a previous work [PLSK11] that first observed this optimization but did not provide an explanation – we fill the above gap, using the results on pseudoinverses given in Chapter 10. Then, we further extend the mathematical framework, introducing pseudoinverses and the isomorphism result for *restricted functions*, which are then used to outline a novel algorithmic optimization for (min,+) and (max,+). We show that these optimizations improve again computation times by several orders of magnitude – however, they do not overlap with the subadditive convolution improvements in Chapter 13.

In Chapter 15, we compare some of the functional features and design choices of *Nancy* and *RTC Toolbox*. We discuss the choice of floating point numerical types and the potential pitfalls of software that use them, then we show, aided by synthetic benchmarks, how anomalies in *RTC Toolbox* suggest that these pitfalls apply there – making the library generally more unstable the larger are the numbers used. We also show, through these benchmarks, that thanks to the rational numerical types *Nancy* does not suffer from the same issues, while the optimizations discussed in the previous chapters allow it to surpass *RTC Toolbox* in many cases – against what the hardware support for floating point numbers would suggest.

Finally, in Chapter 16, we provide some closing remarks and discuss promising future venues of research stemming from this thesis, including on one hand, investigation of algorithmic properties, related to the ones discussed in this thesis, which may also provide similar performance benefits, and on the other hand, further development on the *Nancy* library and its toolset, providing new tools to the research community.

Note on experimental setups

In the following chapters, we discuss our contribution using experimental results. In these experiments, we define a parametrized type curve and then randomly generate such parameters. We use then this set of randomly generated curves to perform operations and compare how they behave with and without our optimizations. These experiments share, in general, the software setup described in detail in Appendix H.1 – we will mention how each experiment may deviate from this when appropriate.

Moreover, we used multiple hardware configurations throughout our experiments. We provide a detailed description of each system in Appendix H.2, and will mention when appropriate which of these systems has been used for a given experiment.

Chapter 8

The *Nancy* Library

The Nancy *software library has been presented in [ZS22], coauthored with Giovanni Stea.*

In this chapter, we discuss software architecture of the *Nancy* library, and the approaches taken while facing the challenge of developing a library that allows, at the same time, for ease of use of existing state of the art results *and* research and development of new ones.

The library is designed with different goals in mind. First, from the perspective of an external user, the interface is supposed to appear minimal and intuitive, and be able to optimize computations by employing, transparently, the best suited algorithms by default. Second, the internal architecture should, on one hand, allow for extensibility via specialization of algorithms, on the other hand, have an intuitive organization that, matching the mathematical model implemented, enables formal verification of said implementation.

Similarly, different objectives guide the choice of programming language and framework. On one hand, the library should be usable in a variety of contexts, from command line applications to fully featured graphical interfaces, as well as interactive environments, such as notebooks, best suited for research applications. Similarly, such usability should not come with excessive learning load for the user, as the syntax to use the library should appear, if not already familiar, easy to learn and based only on the most language features that are commonly understood. On the other hand, the language used for the library should not be overly simplistic, as this could impede the use of advanced techniques for optimizing the library. For example, the language should support ease of parallelization of algorithms, as well as the use of *generator pattern* (a thorough introduction is provided in Appendix A) to reduce active memory utilization while performing algorithms that involve large collections, which are frequent, in practice, in DNC.

In this chapter, we discuss the choices done in *Nancy* with reference to the above goals. We discuss the architecture and implementation through simplified snippets of code. Moreover, we provide an extensive example through the (min,+) convolution, showing how different parts of these operations are implemented and optimized in the layers of the architecture.

8.1 Choice of language and framework

Nancy is coded in C# 11 and .NET 7.0[1], and consists in more than 32k lines of code, including in-code documentation and tests. C# is an object-oriented, statically and strongly typed compiled language. Both build tools and runtime are cross-platform and open source.

As a .NET library, *Nancy* can be used in a variety of context supported by .NET, spanning from console to GUI applications and web services. It can be used also from other languages that can interface with .NET, including F#, a functional programming language, and Visual Basic. Moreover, .NET Interactive Notebooks enables to write and test code using *Nancy* and see its results on a short feedback loop, which is in particular useful for research.

The C# base syntax is very close to other imperative and object-oriented languages such as C++ and Java, thus it can be expected to look familiar to most users. On the other hand, the language offers advanced syntax that is used *within* the library to improve the readability of the code and its performance.

One such feature is LINQ[2], which enables the user to write efficiently algorithms that involve large collections. We provide an example in Listing 8.1. LINQ implements the generator pattern, and together with the `yield` syntax this can be leveraged to minimize memory allocations during computations at a near zero cost in programming effort. How this is achieved is shown in Appendix A, which discusses the pattern at length.

Listing 8.1 Example of LINQ code to get naturals multiples of 6 and less than 100.

```
var list = GetMultiplesOf2()
    .Where(n => n < 100)
    .Where(n => n % 3 == 0)
    .ToList();
```

Moreover, C# classes can be coded so that objects are *immutable*, meaning that any instance method of an object may not change the object itself, rather it will re-

[1]These were released during the writing of this thesis. The version of *Nancy* used for the results discussed here was based on C# 10 and .NET 6.0.

[2]*Language INtegrated Query*.

turn a new modified one.[3] While this technique may appear to inefficiently use memory in some contexts[4], it has the strong advantage of ensuring that each object and operation is *thread safe*, meaning that the same object can be used by multiple algorithms running in parallel.

This can be exploited together with another C# feature, PLINQ (Parallel LINQ). The latter enables one to make LINQ code to run using multiple cores, if available at runtime, as exemplified in Listing 8.2.

Listing 8.2 Example of PLINQ code: the numbers are now checked using multiple cores.

```
var list = GetMultiplesOf2()
    .AsParallel()
    .Where(n => n < 100_000)
    .Where(n => n % 3 == 0)
    .ToList();
```

The main downside of using PLINQ is the risk of high overhead cost if the number of items that can be processed in parallel is low. We avoid this in *Nancy* by using thresholds to decide whether a given query should be executed using standard LINQ or PLINQ. These thresholds were set based on benchmarking results, although we note that such performance results may vary with the algorithm being run, system architecture, memory availability, etc.

In the following, we will mention algorithms in which we exploit this to improve performance, e.g., in the `LowerEnvelope()` algorithm discussed in Section 8.8. From the perspective of a user, though, exploiting the parallelism of *Nancy* is as simple as turning a switch on – or rather, not turning it off. This, and many other optimizations, are in fact controlled with a `ComputationSettings` object that the user can pass, optionally, to most methods of the library to control its behavior.

The `ComputationSettings` class provides many switches for many parts of the library, and is planned to be expanded to provide more control to the user. Being already quite large, we will only mention, when appropriate, the relevant toggles to the discussion at hand. For parallelism, many switches control whether different methods should exploit parallelism or not, and with which thresholds, plus a catch-all `UseParallelism` field that can enable or disable all at once – by default `UseParallelism = true`, so users of the library do not need to do anything to gain from this.

[3]Our definition focuses on immutability from an *external* point of view: the object may cache results to improve future computations.

[4]In these cases, however, one can expect the garbage collector to manage and reclaim the memory as needed.

8.2 Numerical types

As discussed in Part I, the mathematical model focuses on rational numbers, $\mathbb{Q}$. For this library, we chose to avoid using approximate types such as floating point numbers, to avoid the inconsistencies and issues that arise from these approximations and operations such as lcm. In Chapter 15, we further discuss this point and how, we believe, these issues affect the *RTC Toolbox* [WTa].

The native types provided in C# do not include one that accurately represents rational numbers, thus we defined a custom type, `Rational`, including all the operations needed. We provide two implementations of this type, whose choice can be changed with a compiler flag.

The first implementation uses two `long` integers (64 bit) to store the numerator and the denominator, respectively. While this provides a very large set of representable numbers, without any loss of accuracy, one may still find edge cases for which such bit length is not enough, causing an overflow exception at runtime. The `Rational` type uses this implementation if the `LONG_RATIONAL` compile flag is used. Moreover, `LongRational` type provides the same implementation, regardless of compile flag used.

The second implementation uses instead the `BigInteger` type. This type uses an efficient list data structure to use as many bits needed to represent the required result, so that, as long as enough memory is physically available, any integer may be represented. As a non-primitive type, it can be expected to incur in overhead compared to using `long`, even if small numbers are being represented. The `Rational` type uses this implementation if the `BIG_RATIONAL` compile flag is used, while `BigRational` provides it regardless of compile flag used.

In micro-benchmarks, such as the one discussed in Figure 15.1 of Chapter 15, we observed `LongRational` to be up to 5x faster tan `BigRational`. However, in practical use within the library, the difference is usually smaller, and the comparison is complicated by the fact that larger studies tend to push `LongRational` towards overflowing. Since we show that higher performance gains can be achieved without compromises on accuracy and computability, we believe the tradeoff to be in favor of `BigRational`.

Note that, in cases where using `long` produces an overflow, the library will throw an exception and stop computations. Since the two libraries provide the same namespaces and classes, it is sufficient to then change the library loaded to the `BigRational` one and recompile, without further changes to the user code.

The drawback of this approach is that one cannot compile code that uses *both at the same time*. One would need, instead, to use the trickier approach of dynamic (i.e., at runtime) loading of the library.

8.3 Layered architecture

The *Nancy* library is organized in layers which reflect the mathematical model described in Chapter 3. These are:

- The `Curve` layer, which represents functions of $\mathcal{U}$,
- The `Sequence` layer, which represents functions of $\mathcal{U}$ over a finite interval,
- The `Element` layer, which is composed of `Point`s and open `Segment`s.

A schema of the `Curve` class is shown in Listing 8.3. It implements the `representation` of a function $f \in \mathcal{U}$: the `BaseSequence` contains the values of f in the interval $[0, T + d[$, while `PseudoPeriodStart`, `PseudoPeriodLength` and `PseudoPeriodHeight` provide the values of, respectively, T_f, d_f and c_f.

Listing 8.3 Schema of the Curve class.

```
public class Curve
{
    Sequence BaseSequence;
    Rational PseudoPeriodStart;
    Rational PseudoPeriodLength;
    Rational PseudoPeriodHeight;
    [...]
}
```

A schema of the `Sequence` class is shown in Listing 8.4. It implements a *sequence* as defined in Definition 3.6: `Elements` is in fact an ordered set of elements that alternate between points and segments, so that they provide the values of a function f in a finite interval I (which could be, on both sides, either open or closed).

Listing 8.4 Schema of the Sequence class.

```
public class Sequence
{
    List<Element> Elements;
    [...]
}
```

Lastly, the `Element` class is an abstract class, which is implemented by `Point` and `Segment`, as is shown in Listing 8.5. Again, these provide an implementation to the formal definitions provided in Definitions 3.4 and 3.5.

Each class provides a rich API that allows to implement other algorithms. Given an operation to implement, the task is divided between these layers. On one hand,

Listing 8.5 Schema of the Element, Point and Segment classes.

```
public abstract class Element
{
    [...]
}

public class Point : Element
{
    Rational Time;
    Rational Value;
    [...]
}

public class Segment : Element
{
    Rational StartTime;
    Rational EndTime;
    Rational RightLimitAtStartTime;
    Rational Slope;
    [...]
}
```

this has the advantage of establishing clear responsibilities that ease code navigation, maintenance and extensibility; on the other hand, given the nature of implementing a mathematical library, this allows also to focus on the individual operations to verify, formally, their correctness.

In the following sections, we discuss the example of the (min,+) convolution, mentioning all the parts the were designed to work together to efficiently implement the results of [BT07; BT08]. This discussion will be also useful for the following chapters, where we provide improvements to this same algorithm, as well as the implementation of other operations according to the same overall strategy.

8.4 Implemented operators

The main operators that are implemented by the library are

- `Addition` and `Subtraction`
- `Minimum` and `Maximum`
- (min,+) `Convolution` and `Deconvolution`
- `MaxPlusConvolution` and `MaxPlusDeconvolution`
- `SubAdditiveClosure`, `SuperadditiveClosure`

- `LowerPseudoInverse`, `UpperPseudoInverse`
- `Composition`

For most of the above, the algorithm is either discussed in [BT08; BBL18] or can be derived along the same steps (e.g., `Maximum` and `MaxPlusConvolution`). As already mentioned, of these we will discuss in depth only the (min,+) `Convolution` operator, since it is relevant both to present the architecture and for the algorithmic improvements discussed in the following chapters.

The algorithms for pseudoinverses and composition are instead novel results, discussed in Chapters 10 and 11.

8.4.1 Note on plain and ultimately plain functions

As mentioned in Section 3.8, for some of these operators, such as minimum and convolution, [BT08] shows that the operands being in $\mathcal{U}$, alone, is not *sufficient* to assert that the result is again in $\mathcal{U}$. A sufficient condition is instead obtained under additional hypotheses such as operands being *plain* or *ultimately plain*. On the other hand, these are not *necessary*, i.e., one can construct a case where operands that are not *ultimately plain* still produce a result in $\mathcal{U}$.

Lacking, to the best of our knowledge, a set of conditions that are both *sufficient and necessary*, the library leaves to the user the task to check that the operation is meaningful. To aid them with this task, the library implements the `IsPlain` and `Is⌋ UltimatelyPlain` checks.

8.5 The *Cut* algorithm

An important part of the $\mathcal{U}$ mathematical model is that UPP functions allow us to store the representation of a function f over a finite interval $\left[0, T_f + d_f\right[$, and to derive the values for any t in $\left[T_f + d_f, +\infty\right[$ only when needed, using Equation (3.1). While intuitively simple, it is worth taking some time discussing how this can be efficiently implemented, as this also affects how the rest of the library may be built on top of it.

Assume that an algorithm needs the values of f in $[t_a, t_b[$, where $t_a < T_f + d_f$ and $T_f + 2 \cdot d_f < t_b < T_f + 3 \cdot d_f$. With reference to Figure 8.1, we can regard f as being composed of a *transient* and a *period* (already stored in memory, as `BaseSeque⌋ nce`), followed by an infinite amount of *period replicas*, to be computed on the fly as needed.

A first implementation may consider a method, say `GetPeriodReplica(int i)`, that returns the elements that compose the i-th period replica. Then one can bound the latest i that will be used by computing $i_{\max} = \left\lceil \frac{t_b - T_f}{d_f} \right\rceil$, and construct a list containing

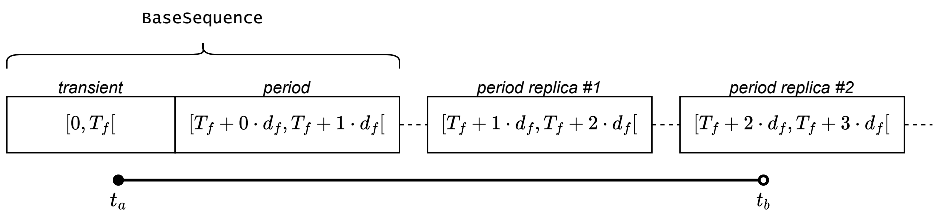

BaseSequence
transient
period
period replica #1
period replica #2
$[0, T_f[$
$[T_f + 0 \cdot d_f, T_f + 1 \cdot d_f[$
$[T_f + 1 \cdot d_f, T_f + 2 \cdot d_f[$
$[T_f + 2 \cdot d_f, T_f + 3 \cdot d_f[$
t_a
t_b

Figure 8.1: View of a cut of f and the required *period replicas*.

all elements from the `BaseSequence` and all period replicas $i = 1 \dots i_{\max}$. Then, a filtering step (assume it is implemented by `Sequence.Cut()`) can remove the parts for $t < t_a$ and $t \geq t_b$ from said list, the result from this step can then be returned. This process is summarized in the (simplified) code of Listing 8.6.[5]

Listing 8.6 Scheme of the `Curve.Cut()` method

```
Sequence Cut(Rational t_a, Rational t_b) {
    var i_max = Math.Ceil( (t_b - this.T) / this.d );
    var elements = new List<Element> { this.BaseSequence.Elements };
    for(int i = 1; i <= i_max; i++) {
        elements.AddRange( GetPeriodReplica(i) );
    }
    var sequence = new Sequence(elements);
    return sequence.Cut(t_a, t_b);
}
```

There are some inefficiencies in this approach. The first obvious one is that performing the filtering *after* allocating all elements involves more active memory utilization, which can be detrimental due to the high number of elements than may be involved in some computations. The second, less obvious, one is that many algorithms require elements *up to* t_b, but may, for many reasons, terminate earlier and never process elements after some t – thus their allocation is unnecessary in the first place.

Both can be addressed by implementing `Cut` as a generator method, that returns elements one at a time and allocates them only when necessary. To be efficient, this requires also `GetPeriodReplica()` to be a generator method, so that we only retrieve the parts of replica $i_{\max}$ that are within t_b. This is summarized in Listing 8.7, where we assume for simplicity that t_a and t_b are breakpoints (Definition 3.7).

[5]Given two `Lists` `a` and `b`, `a.AddRange(b)` appends all elements of `b` to `a`.

Listing 8.7 Scheme of the `Curve.CutAsEnumerable()` method

```
IEnumerable<Element> CutAsEnumerable(Rational t_a, Rational t_b) {
    var i_max = Math.Ceil( (t_b - this.T) / this.d );
    foreach(var element in this.BaseSequence.Elements) {
        if (element.StartTime < t_a)
            continue;
        else
            yield return element;
    }
    for(int i = 1; i <= i_max; i++) {
        foreach(element in GetPeriodReplica(i)) {
            if (element.StartTime < t_b)
                yield return element;
            else
                yield break;
        }
    }
}
```

On the other hand, this approach of allocating one replica at a time may not take advantage of parallelization. In fact, it the number of replicas needed is large enough, it may be useful to parallelize their computation. To do this efficiently, one would have to provide an algorithm that mixes the two strategies, where batches of elements are allocated in advance and in parallel, yet returned only as necessary.

Finally, the semantics of the two methods, `Cut` and `CutAsEnumerable`, are clearly different: most users may still expect a method that simply returns a `Sequence`, rather than an `IEnumerable<Element>`, which will require an introduction on generators to most. Hence, *Nancy* provides *both*.

8.6 Implementing the *by-curve* (min,+) convolution

We discuss now how the (min,+) convolution, presented in Chapter 3, can be implemented. We assume for simplicity, that f and g are neither UA nor UI, and such that $\rho_f \neq \rho_g$, i.e., we avoid those cases where optimization opportunities – which we do take in *Nancy* – would complicate our discussion. Another assumption is that this convolution is *well-defined*, i.e., there are no t_1, t_2 such that $f(t_1) = +\infty$ and $g(t_2) = -\infty$, or vice versa [BT08, p. 7] – in such case, our implementation will throw a runtime exception.

As discussed in Section 3.4, to compute the (min,+) convolution of two generic UPP functions we need to decompose the convolution into four terms, compute them separately, and then compute their minimum. This is summarized in Listing 8.8.

Listing 8.8 Scheme of the `Curve.Convolution()` method

```
Curve Convolution(Curve f, Curve g)
{
    var h_tt = ConvolutionTransientTransient(f, g);
    var h_tp = ConvolutionTransientPeriodic(f, g);
    var h_pt = ConvolutionTransientPeriodic(g, f);
    var h_pp = ConvolutionPeriodicPeriodic(f, g);
    return Minimum(h_tt, h_tp, h_pt, h_pp);
}
```

We mention in particular the algorithm for the fourth term, i.e., the convolution between periodic parts, as it is also the focus of the optimizations discussed in Chapter 14. The method uses the results of Proposition 3.15 to compute $f_p^{\wedge} \otimes g_p^{\wedge}$. We provide a first implementation of it in Listing 8.9, which we then discuss for improvements.

Listing 8.9 Scheme of the `Curve.ConvolutionPeriodicPeriodic()` method

```
Curve ConvolutionPeriodicPeriodic(Curve f, Curve g)
{
    var d = Rational.LeastCommonMultiple(f.d, g.d);
    var c = d * Rational.Min(f.rho, g.rho);
    var T = f.T + g.T + d;
    var fCut = f.Cut(f.T, f.T + 2 * d);
    var gCut = g.Cut(g.T, g.T + 2 * d);
    var seq = Sequence.Convolution(fCut, gCut)
    var resultSeq = seq
        .Cut(f.T + g.T, f.T + g.T + 2 * d)
        .PrependWithPlusInfinity(0, f.T + g.T);
    return new Curve(resultSeq, T, d, c);
}
```

We stress the use of `Sequence.Cut` in this code: in fact, note that from the *by-sequence* convolution, we may obtain a larger sequence than that we are interested in. In fact, consider the case of an element e_f whose support is $\subset \left[T_f + d, T_f + 2 \cdot d\right[$, and similarly for e_g whose support is $\subset \left[T_g + d, T_g + 2 \cdot d\right[$. While, as shown in Proposition 3.16, these elements are both needed for the correct computation of the convolution, their convolution $e_f \otimes e_g$ has support I where $I \cap \left[T_f + T_g, T_f + T_g + 2 \cdot d\right[= \emptyset$. Thus, due to $e_f \otimes e_g$ and similar cases, the *by-sequence* convolution may have a larger support than needed, and needs to be filtered according to the interval that, via the formal results, we are sure to be correct.

On the other hand, we note that computing $e_f \otimes e_g$ is unnecessary in the first place, and we can optimize the *by-sequence* convolution by skipping this and other similar cases. We discuss this in the following section.

8.7 Implementing the *by-sequence* (min,+) convolution

As described formally in Section 3.4.2, in the *by-sequence* convolution we decompose each sequence into their constituting elements, and compute the set of *by-element* convolutions (i.e., point with point, point with segment, segment with segment). These are discussed thoroughly in [BT08, Lemma 2-4] and, being trivial to implement, we will omit their discussion. Then, given E, the set of elements resulting from these *by-element* convolutions, we compute their lower envelope, which concludes the algorithm. We provide a first version of this in the following.

Listing 8.10 Scheme of the `Sequence.Convolution()` method

```
Sequence Convolution(Sequence sf, Sequence sg)
{
    var convolutionElements = new List<Element>();
    foreach(var e_f in sf){
        foreach(var e_g in sg){
            convolutionElements.AddRange(Element.Convolution(e_f, e_g));
        }
    }
    var le = LowerEnvelope(convolutionElements);
    return le;
}
```

The code above will compute the *by-element* convolution resulting from *every* pair of elements from sequences sf and sg. However, as we mentioned, there are many cases where we can check *a priori* whether the convolution of a given pair of elements is of any interest for the result. We thus restructure the code in order to support such instances where the pairs need filtering, and we do so taking advantage of the generator pattern. In Listing 8.11, we implement two of these *filters*, leaving out a pair of elements when either is equal to $+\infty$, or when the convolution between them ends up outside the interval of interest (as discussed in the previous section).[6]

[6]In C#, syntax `(Element ef, Element eg)` denotes a *tuple*. Tuples can be seen as a shorthand for classes whose only purpose is to group values together.

Listing 8.11 Scheme of the `Sequence.Convolution()` method, restructured to use filters

```
Sequence Convolution(Sequence sf, Sequence sg, Rational cutEnd = Rational.PlusInfinity) {
    var convolutionPairs = GetFilteredElementPairs(sf, sg, cutEnd);
    var convolutionElements = new List<Element>();
    foreach(var pair in convolutionPairs)
        convolutionElements.AddRange(Element.Convolution(pair.ef, pair.eg));
    var le = LowerEnvelope(convolutionElements);
    return le;
}

IEnumerable<(Element ef, Element eg)> GetFilteredElementPairs(
    Sequence sf, Sequence sg, Rational cutEnd) {
    return GetElementPairs(sf, sg)
        .Where(pair => pair.ef.IsFinite && pair.eg.IsFinite)
        .Where(pair => pair.ef.StartTime + pair.eg.StartTime < cutEnd);
}

IEnumerable<(Element ef, Element eg)> GetElementPairs(Sequence sf, Sequence sg) {
    foreach(var ef in sf)
        foreach(var eg in sg)
            yield return (ef, eg);
}
```

In this version, `Sequence.Convolution()` has a third, optional argument, with which the caller can specify the end of its interval of interest – so that unuseful convolutions may not be computed. This structure can be efficiently extended to add more cases for which *by-element* convolutions may be removed – as discuss in Chapter 14.

The process of computing the *by-element* convolutions is also highly parallelizable, as it is a (usually) large number of operations that are independent of each other and without side effects. Since the *number* of pairs is crucial for the tradeoff between serial and parallel operations, we do a counting step beforehand, using the LINQ `LongCount()` method.

```
var pairsCount = GetFilteredElementPairs(sf, sg, cutEnd).LongCount();
```

Note that this line – being based on generators – has no allocation cost and, in our benchmarks, takes very little time compared to the runtime gain between the two methods. In *Nancy*, the use of a parallel *by-sequence* convolution is controlled by the settings `UseParallelConvolution` and `ConvolutionParallelizationThreshold`, i.e., the parallel algorithm is used only when `pairsCount` is above the threshold.

The *by-sequence* convolution may easily produce, in our experiments with more complex convolutions, millions of elements. Storing them in memory at the same time to compute their lower envelope may be taxing for the physical memory, and we recall that on modern systems performance may degrade by orders of magnitude when the operating system is left with little available memory left.

However, we note that, in our experience, the large majority of elements of E do not end up being part of the lower envelope, i.e., $n(E) \gg n(\bigwedge E)$. Moreover, the lower envelope of set E can be *partitioned*, as $\bigwedge E = (\bigwedge E_1) \wedge \cdots \wedge (\bigwedge E_n)$, where each E_i is a partition of E with more manageable cardinality. We can exploit, for this, the fact that the `pairsCount` can be computed cheaply and without allocating the entire set at once – i.e., we can check beforehand if, even with filters in mind, we risk saturating the memory. If `UseConvolutionPartitioning` is `true` and the count is above `ConvolutionPartitioningThreshold`, we proceed with the partitioned convolution algorithm. We use the LINQ `Chunk()` method to allocate and retrieve the pairs by partitions (or *chunks*) of `ConvolutionPartitioningThreshold`. For each of these partitions we proceed with the usual algorithm, computing the *by-element* convolutions and then their lower envelope – which can be parallelized, using the same settings discussed just before. Then, we compute the minimum of these partial lower envelopes – which can also be parallelized, using `UseParallelListLowerEnvelope`. In the following section, we close the only remaining gap, discussing the implementation of the lower envelope algorithm.

8.8 Lower envelope algorithm for (min,+) convolution

In this subsection we discuss the algorithm to compute the lower envelope of a set E of elements resulting from *by-element* convolutions. Such set is in general not ordered and with varying overlap patterns, as shown in Figure 8.2a. Hence, differently from the case of the minimum, where we have two perfectly overlapping sequences, it would be inefficient to attempt to compute the lower envelope by direct comparison – given $n(E)$ the number of elements, this would have $\mathcal{O}\left(n(E)^2\right)$ complexity.

Instead, we obtain $\mathcal{O}\left(n(E) \cdot \log(n(E))\right)$ complexity by grouping the elements in E by *intervals of overlap*, and compute the lower envelope as the juxtaposition of *per-interval* lower envelopes. The result of such process, which we describe in detail in the following, is exemplified in Figure 8.2b. Considering the interval $]5,6[$, the lower envelope over such interval is equal to the lower envelope of only those segments whose support contains $]5,6[$. The same can be said for point-sized intervals such as $\{3\}$ and $\{6\}$.

Definition 8.1. Let e_1 and e_2 be *elements* with, respectively, support I_1 and I_2. We say that they share *interval of overlap* I if $I = I_1 \cap I_2 \neq \emptyset$, and that e_1 and e_2 *belong* to I.[7] We say I is *point-sized* if it is of the form $\{t\}$; we say instead it is *segment-sized* if it is of the form $]t_a, t_b[$.

[7] An element, more precisely a segment, may *belong* to more than one *interval of overlap*.

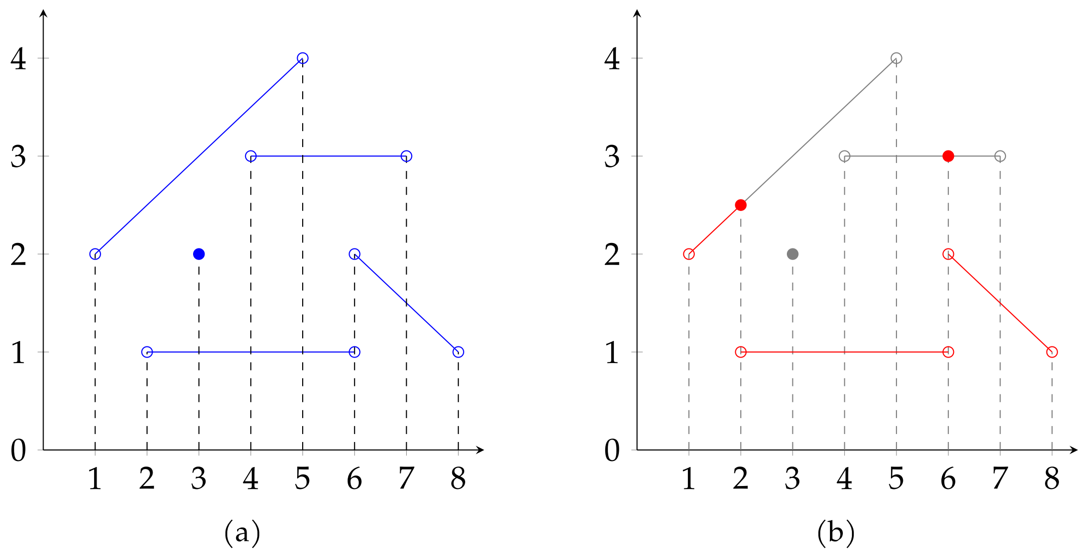


Figure 8.2: Intervals for a set of elements (a) and their lower envelope (b).

Note that, given I_i the support of element i, $i = 1, \ldots, n$, the set of *intervals of overlap* forms a partition of their union $\bigcup_{i=1\ldots n} I_i$. Then, the process is as follows

- collect the start and end times of all the elements in E and order them. Let the result be $t_0, t_1, \ldots, t_N$;
- derive *point-* and *segment-sized* intervals for and between each of these times, i.e., $\{t_0\},]t_0, t_1[, \{t_1\}, \ldots;$
- associate to each of the above interval the set of elements that *belong* to it, initially empty;
- for each element $e \in E$, find all the intervals I belongs to, and add e to their lists.

We underline that the element-interval relationship is many-to-many: the same element may span multiple intervals, and an interval may include several elements. Afterwards, we compute the lower envelope over each interval I, and we concatenate these to obtain the overall lower envelope of E.
As for the algorithm costs, we note that

1. *finding* which intervals an element belongs to is $\mathcal{O}(n \cdot \log(n))$, where n is the number of intervals, if one uses an interval tree;
2. *inserting* an element in the lists of *all the intervals* it belongs to, instead, depends on the number of intervals an element belongs to (something which we show to be highly variable in a few lines), and is $\mathcal{O}(n)$ in a worst case;

3. computing the per-interval lower envelope of I costs $\mathcal{O}(m)$ if I is point-sized, $\mathcal{O}(m \cdot \log(m))$ if segment-sized, where m is the cardinality of E_I;

4. the concatenation of the per-interval results is $\mathcal{O}(n)$.

Of the above steps, we note that steps 2 and 3 are independent of the number of elements, but instead depend on how much overlap there is between them. In fact, the more overlap there is between the elements of E, the higher the cost of this algorithm is. Some of the overlaps – actually, most – will not yield segments that end up being part of the lower envelope. This phenomenon can lead to large differences in the *actual* computation time between lower envelopes of sets with similar cardinality. We highlight one such case in Section 13.2.

This algorithm is implemented in *Nancy* as the `Sequence.LowerEnvelope()` static method, which uses the `Interval` and `InteralTree` classes internally. An `Interval` represents an *interval of overlap* and the set of `Element`s that belong to it, while `IntervalTree` implements an efficient and thread safe query method to obtain the set of `Interval`s an `Element` overlaps with.

Given the set of elements, `elements`, `Sequence.LowerEnvelope()` does the following:

1. uses the `Interval.ComputeIntervals()` static method to get the ordered set of *interval of overlap*, each with the elements that belong to it;

2. for each *interval*, it uses `interval.LowerEnvelope()` to obtain the per-interval lower envelope. This is done independently for each `interval`, hence it can be parallelized;

3. returns the lower envelope by concatenating, in order, the per-interval lower envelopes.

Moreover, the `Interval.ComputeIntervals()` static method does the following.

1. collects the set `times` of start and end times of all elements, ordered and distinct. This can be parallelized using PLINQ,

2. instantiates the set `intervals` of `Interval` based on this set, and an `IntevalTree` over said set,

3. for each *element*, it uses the `IntervalTree` to obtain the set of `Interval`s it overlaps with; and then adds the element to their respective sets,

4. returns the set `intevals`.

Step 3 of the above algorithm is composed of independent operations, although parallelizing it can be tricky. In fact, while the queries to `IntervalTree` can be run in parallel, appending elements to the collection of each `Interval` requires a thread-safe collection (e.g., `ConcurrentBag<T>`), which is much slower than a `List<T>` when not actually used concurrently. The alternative approach we take is instead to expand the result of the `IntervalTree`, i.e., from each `(Element, List<Interval>)` result we get a `List<(element, interval)>`. Collecting together all these overlaps, we can then group them by `inteval`, obtaining then a set of `(interval, List<Element>)` tuples. Then the insertion of these `Element`s into the corresponding `interval`s can be executed in parallel and thread-safely using the performant `List<T>` as backing storage.

We remark that the tradeoff between a) concurrent insertion using `ConcurrentBag<T>` and b) doing the extra grouping work for thread safe parallel insertion using `List<T>` depends, other than the number of elements and overlaps, also on the hardware and its support for efficient concurrent data structures.

The parallelization of `Sequence.LowerEnvelope()` is controlled by the setting `UseParallelLowerEnvelope`, while the parallelization of `Interval.ComputeIntervals()` is controlled by settings `UseParallelComputeIntervals` and `settings.ParallelComputeIntervalsThreshold` (the algorithm is not parallelized unless the number of elements is above the threshold).

8.9 Inheritance and specialization of algorithms

As mentioned, the algorithms need to be *specializable*, i.e., to allow for extension with more efficient algorithms for particular cases (e.g., when additional hypotheses on the operands are available). We do this in two ways: *within* the standard algorithm and through *subclasses*.

An example of the first strategy are the optimizations for Ultimately Affine functions. In this case, the main source of optimization is that there are no breakpoints within the pseudo-periodic part of the function, and, more importantly, the period length is immaterial. Thus, whenever we are computing the hyperperiod of two functions where one is UA, we set their period length to be 1, i.e., the identity element w.r.t. the lcm. This property is simple and cheap to verify, while the algorithms are mostly unchanged, aside from the hyperperiod optimization mentioned above. Thus, reaping the benefits of Ultimately Affine functions requires no extra code from the user.

The *subclassing* approach is, instead, meant for the different case of properties that are harder to verify on-the-fly, that alter algorithms significantly, or that are simply of particular significance from a modeling perspective.

An example of the first two is the `SubAdditiveCurve` class, which is used to mark curves that are subadditive and have $f(0) = 0$. Computing the (min,+) convolution of such curves can then benefit from the optimizations discussed in Chapter 13. Similarly, `ConcaveCurve` marks curves that are concave and have $f(0) = 0$, for which is known that $f \otimes g = f \wedge g$ ([BBL18, Prop. 3.12]). These optimizations are then implemented by *overriding* the `Curve.Convolution()` method, e.g., with `SubadditiveCurve.Convolution()`, taking into account these properties to try and avoid computing the convolution at all.

However, verifying that a function is subadditive, as discussed in detail in Section 9.3, comes at a significant computational cost. Testing for concavity is cheaper (a linear scan), but would still add a significant overhead. Thus, we leave to the user to check these properties when deemed appropriate, and manually mark known properties of functions using the corresponding subclass. Similar classes, with similar optimizations and concerns, are available for superadditive and convex functions.

We clarify the above with some examples. Suppose that we compute a curve f that is *known* to be subadditive. We can make this explicit using the following code:

```
// if doTest is true, the property is tested,
// and an exception is thrown if the curve is not actually subadditive
// if doTest is false, the library 'trusts' the user and skips checking
var f = new SubAdditiveCurve(/* parameters */, doTest: false);
```

The above is done automatically when performing computations that are known to produce subadditive functions, e.g., the subadditive closure or convolution between subadditive functions.

```
// f will be automatically marked as SubAdditiveCurve
var f = (/** some other curve **/).SubAdditiveClosure();
```

Consider now, instead, that we have a curve `g` that *might* be subadditive and, if that is the case, we desire to exploit this property for optimizations when we compute its convolution with `f`. Then we may use the following:

```
var g = /* some code */;
if(g.ValueAt(0) == 0 && g.IsSubAdditive) { // may be an expensive check
    g = new SubAdditiveCurve(g, doTest: false); // mark it as subadditive using the subclass
}

// the following convolution may use the standard algorithm or the optimized one,
// depending on the result of the above check
var h = Curve.Convolution(f, g);
```

On the other hand, classes such as `RateLatencyServiceCurve` serve the purpose of simplifying the construction and management of curves that are common in DNC studies. Moreover, one can provide overrides to make the operator result more specific. For example, the addition of a `RateLatencyServiceCurve`, $\beta_{R,\theta}$, with a `ConstantCurve` [8] , W, yields a `RaisedRateLatencyServiceCurve`, $\beta_{R,\theta} + W$. Then we exploit the fact that, as discussed in Section 3.6, for this expression we can compute the subadditive

[8]As demonstrated later, when performing the addition of a curve with an `int`, the latter is implicitly cast into a `ConstantCurve`.

Listing 8.12 Example of unit test used in Nancy.

```
[Fact]
public void FlowControlExpressionTest()
{
    var f = (new RateLatencyServiceCurve(2, 1) + 1).SubAdditiveClosure();
    // test properties
    Assert.False(curve.IsContinuous); // there is a right-discontinuity at 0
    Assert.False(curve.IsRightContinuous);
    Assert.True(curve.IsLeftContinuous);
    Assert.True(curve.IsContinuousExceptOrigin);
    Assert.True(curve.IsNonDecreasing);
    Assert.True(curve.IsSubAdditive);
    // test some function values
    Assert.Equal(0, f.ValueAt(0));
    Assert.Equal(1, f.RightLimitAt(0));
    Assert.Equal(1, f.ValueAt(2));
    Assert.Equal(2, f.ValueAt(3));
}
```

closure in closed form. In fact, `RaisedRateLatencyServiceCurve` overrides the `Curve.SubAdditiveClosure()` method, and computes the result using said closed form rather than the generic $\mathcal{NP}$-hard algorithm.

The above means that a user can compute the result of expressions such as $\overline{\beta_{2,1} + 1}$ writing code that intuitively resembles it *and* is efficient to execute, as shown below.

```
// this code is O(1)
var b = new RateLatencyServiceCurve(2, 1);
var f = (b + 1).SubAdditiveClosure();
```

8.10 Testing

For a software implementation, it is important to have a sound testing procedure to a) catch bugs in new features *earlier*, and b) avoid creating new ones in existing features, i.e., *regressions*. *Nancy* comes with a large library of tests, that have been collected during its development and use to verify its correctness and deal with edge cases. These tests are collected in `Nancy.Tests`, using the *XUnit* unit test framework.

For example, one may check that the code closing the previous section yields the correct the result. Using the handy properties and methods discussed in Chapter 9, one can easily check that the object has the expected mathematical properties, i.e., those of $\overline{\beta_{2,1} + 1}$. The code for such test would then look as in Listing 8.12. When run, the *XUnit* framework will then execute these test, and report any assertion that is not verified.

There are many ways we used to gather the tests that make up the current collection. The first are hand-picked examples whose computation is simple enough to perform using pen and paper, as is indeed the case for the example in Listing 8.12. Then we have the algebraic equivalences, i.e., relationships between the operators that we verify to hold within *Nancy* as it is expected from the formal proofs – e.g., Lemma 3.26. Lastly, we have the use cases that came from practical use of the library, which highlighted edge cases or outright bugs in the past. In those cases, we would first isolate the issue into one or more failing tests. Fixing the root issue would then naturally turn those tests into passing ones and, afterwards, they remained useful to make sure future changes do not revert the behavior into an incorrect one – what is usually called a *regression*.

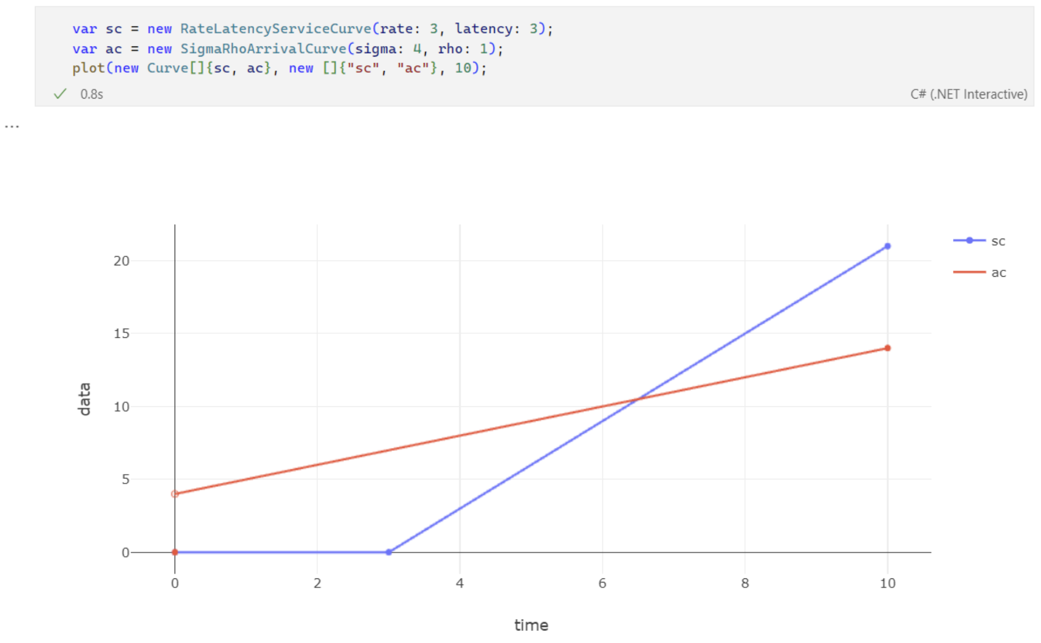
In the approach mentioned above, we manually checked the results of the library looking for inconsistencies and, ultimately, the root causes of the issues at hand. However, recent results [RRB21; BRD22] show that mathematical properties can be leveraged to build a *verifier* of tools, i.e., a tool that can be used to test if a given result is correct. It would then be valuable to apply such automated verification on the results of this library, to more easily identify issues and further build confidence on the correctness of the implementation.

8.11 Publication

The *Nancy* library is distributed both as source code, on GitHub [ZSd], and as pre-compiled binaries, on NuGet [ZSc]. On the latter, it is distributed in two forms, distinguished by numerical type: the `Unipi.Nancy` package uses the `BigInteger` implementation, while the `Unipi.Nancy.LongRational` package uses the `long` one. It is provided with the MIT license, a standard license for open source that, differently from free software licenses such as the GPL, allows for wide adoption within commercial and industrial contexts.

In the publication process of the library, we adopted many principles and practices that, in the last decade, have become the standard for quality of open source software. Using GitHub Actions, we set up a CI/CD pipeline that, for any new commit marking a release, will automatically run the tests, build the packages and publish them on NuGet, minimizing the attrition and manual actions required to bring the latest changes to the users. A similar process is followed for the documentation website, whose contents are built directly from the in-code comments of the library.[9]

[9]While many of the tools used to build the website were already available, the process to extract the documentation and format it into Markdown required some coding, an effort in tooling that we plan to share, again, as open source.



```
var sc = new RateLatencyServiceCurve(rate: 3, latency: 3);
var ac = new SigmaRhoArrivalCurve(sigma: 4, rho: 1);
plot(new Curve[]{sc, ac}, new []{"sc", "ac"}, 10);
```

Figure 8.3: Example of use of notebooks, where the call `plot(...)` generates the image below the notebook cell.

8.12 Support for notebooks

Lastly, we mention the work to support *Nancy* within .NET Notebooks. For research, it is particularly useful to be able to quickly experiment with expressions and *see* what their results look like, hence we sought an environment that allowed us just that, with the least friction possible.

.NET Notebooks (like other forms of *notebooks* that have surfaced in the past decade) allow one to write and execute code within an interactive environment. Thanks to the addition of `plot()` function – which we provide via our notebook templates – one can also plot the curves. Of course this would be incomplete without handy methods to "poke" the curve for algebraic properties and values, as well as the ability to retrieve high-quality plots for scientific publication – we discuss these, among other extensions, in the following chapter.

We wish to emphasize that, in our experience, this feature was critical for coming up with, or fixing the issues of, many of the results we discuss within this thesis, or at the very least, to reduce dramatically the time required to do so.

Chapter 9

Useful Extensions

In this chapter, we present some smaller results that – we believe – are of interest towards the implementation of a software library, yet by themselves too small to deserve publication (compared, e.g., to the results discussed in the following chapters). To the best of our knowledge, these results and algorithms have not been discussed in the literature, yet we believe they are easy to derive from, e.g., the model provided in [BT07; BT08; BBL18]. These are both useful to the purpose of implementing the other results discussed in this thesis, but also to enrich the API provided to the user – in our own experience, having these available has far improved the research experience and productivity.

9.1 Sampling at or near a time t

Often, the first step into *debugging* a mathematical object is to observe its values. While most algorithms, as discussed here, are usually focused on processing *sets* of values, too much information is usually overbearing to a user. Moreover, it is easier to write *tests* for the software if one can sample the result at known points and compare with its expected value computed by pen-and-paper means.

The instance methods `ValueAt()`, `RightLimitAt()` and `LeftLimitAt()` provide this functionality across all layers. The most interesting one is the `Sequence` layer: here the algorithm becomes a search for the `Segment` or `Point` whose support contains the required t. This is implemented as a binary search. Given a sequence `s` and the current index range `[a, b]`, the algorithm checks the elements `s[a]`, `s[(a+b)/2]` and `s[b]` to decide its next index range, halving the search range until the target is found.

At the `Curve` layer, the algorithm only checks if $t < T + d$ or otherwise, to find the least initial range of search – if $[0, T + d[$ the search uses the `baseSequence`, otherwise the proper pseudo-period replica must be computed and used. In any case, let S be the sequence within which we search (either the base sequence or a pseudo-period replica), the algorithmic complexity is $\mathcal{O}(\log n(S))$.

The alert reader may note that, given how the search works, the same algorithm can be (and is) used to implement `GetSegmentBefore()`, `GetElementAt()` and `GetSegmentAfter()`.

9.2 Equivalence of two *curves*

As mentioned in Chapter 3, a function f of $\mathcal{U}$ may have multiple *equivalent representations*. For example, given $R_1 = (S, T, d, c)$, one can easily derive that $R_2 = (S', T + 5, 2d, 2c)$, with $S' = Cut(R_1, 0, T + 5 + 2d)$, is another valid, if inefficient, representation of the same function.

It is then useful to have a `Curve.Equivalent()` method that tells whether two different objects, say `c1` and `c2`, are equivalent representations of the same function. The method is implemented according to the following simple result.

Lemma 9.1. *Let f and g be functions of $\mathcal{U}$. Then, $f = g$ if and only if*

$$(f - g)(t) = 0 \; \forall t \geq 0.$$

Since the subtraction operator is UPP from $\max\left(T_f, T_g\right)$ with period length $\text{lcm}\left(d_f, d_g\right)$ (Proposition 3.20), we derive that for the above test it is sufficient to limit the comparison to $[0, M[$, $M := \max\left(T_f, T_g\right) + \text{lcm}\left(d_f, d_g\right)$. Thus, given `sf = f.Cut(0, M)` and `sg = g.Cut(0, M)`, we can iterate over the two comparing elements one by one, and return `true` if we keep finding equivalent elements until we reach the end of both at the same time, `false` in any other case. Lastly, this is an example of an operator that can greatly benefit from the *generator pattern*, using `Curve.CutAsEnumerable()` to improve its efficiency.

9.3 Properties: continuity, subadditivity, etc.

It is useful to distinguish, for the purpose of an efficient implementation, those properties of a function f that may be checked via a single, left-to-right (or vice versa) scan of its representation or an extended view of it, from more complex ones such as sub- and superadditivity. Checks such as `Curve.NonDecreasing`, `Curve.IsLeftContinuous` can be trivially implemented as linear scans that check if the representation violates or not the properties in $[0, T + d]$. In fact, we are guaranteed, via the UPP property, that the represented function f will not violate them unless it does within such range.

A more complex matter is instead to check sub- and superadditivity. For those, we use the following results – we wish to thank Paul Nikolaus for the help in their formalization.

Definition 9.2. Let f be a function of $\mathcal{U}$. Then we define f° as

$$f^\circ(t) := \begin{cases} 0, & \text{if } t = 0, \\ f(t), & \text{otherwise.} \end{cases}$$

Lemma 9.3. *Let f be a function of $\mathcal{U}$ such that $f(0) \geq 0$. Then, f is subadditive if and only if $f^\circ = f^\circ \otimes f^\circ$.*

Lemma 9.4. *Let f be a function of $\mathcal{U}$ such that $f(0) \leq 0$. Then, f is superadditive if and only if $f^\circ = f^\circ \overline{\otimes} f^\circ$.*

We provide proofs in Appendix G.2. These properties allow one to get very useful insights for a DNC study using f. For example, we show in Chapter 13 how the (min,+) convolution of subadditive functions can benefit from an improved algorithm with large performance improvements. We expect that a symmetrical result should hold for the (max,+) convolution of superadditive functions. We have not had time to investigate this, though, and we leave it for future work. Moreover, works such as [GY13; LBS16; LBSGY17] provide an improved algorithm for studying end-to-end delays of tandems involving superadditive service curves.

The properties implemented in the library cover many use cases. We list here some of them:

- `IsFinite`
- `IsPlusInfinite`
- `IsMinusInfinite`
- `IsContinuous`
- `IsLeftContinuous`
- `IsRightContinuous`
- `IsNonNegative`
- `IsNonDecreasing`
- `IsUltimatelyInfinite`
- `IsUltimatelyPlain`
- `IsPlain`
- `IsUltimatelyAffine`
- `IsUltimatelyConstant`
- `IsConcave`
- `IsConvex`

Similarly, the library provides continuity checks at a given t, i.e.,

- `IsContinuousAt`
- `IsLeftContinuousAt`
- `IsRightContinuousAt`

9.4 Closures: non-negative, non-decreasing, etc.

Another useful extension to the toolbox is the direct support for closures such as non-negative and non-decreasing ones. These are operators used in literature to ensure that, even if partial results may have negative parts or decreasing segments, we can reconstruct from them regular non-negative and non-decreasing functions.

As reported in [BBL18, p. 45], the non-negative closure can be implemented as a simple maximum between the function f and a constant zero function, i.e., $f \vee 0$, while the non-decreasing closure can be computed using the (max,+) convolution with a constant zero function, i.e., $f \overline{\otimes} 0$. In *Nancy*, we provide for the latter a more efficient implementation, which only considers the components of said (max,+) convolution that affect the result. Moreover, this provides a non-decreasing function that is *larger* than the output. For this reason, we distinguish and implement the *upper* non-decreasing closure and the *lower* non-decreasing one. Similarly, the library provides left-continuous and right-continuous closures. The methods corresponding to the above are

- `ToUpperNonDecreasing`
- `ToLowerNonDecreasing`
- `ToLeftContinuous`
- `ToRightContinuous`
- `ToNonNegative`

9.5 Formatting as JSON, C#, TikZ

In our research experience, it has proven valuable to be able to move or replicate a result between different contexts, be it to continue testing an intermediate result of a process, to derive a visualization with a different tool, to export a neat result from notebook experimentation into a figure in a scientific article. This process can be greatly improved by appropriate software support, as we discuss here.

JSON: cross-language data representation

In all public classes of *Nancy*, the `ToString()` method will provide a JSON representation of the object, while the `FromJson()` static method will construct an object from its JSON representation. JSON is a human-readable text-based data representation format that, inspired by the JavaScript syntax, has expanded to be a standard for data sharing between applications due its language agnostic nature, widely available support, and human readability.

Thus, this feature brings the important benefit of a native entry point to build a cross-application and cross-language workflow, that integrates *Nancy* with other tools.

C# code: export objects for further study

In all public classes of *Nancy*, the `ToCodeString()` method will provide a string with C# code that can be used to build the object. The target use case for this feature is retrieving an intermediate result from an application (for example, through a debugger instance), so that the same result can be then reconstructed and studied in a .NET Notebook or a test.

TikZ: high quality plots for UPP curves

It would be surely unnecessary to justify, in a scientific work such as this thesis, the value of proper visualization using figures to clearly show a result to the reader. TikZ is a native LaTeX tool to produce high quality figures, however its high complexity can quickly become detrimental towards such goal. Although its text-based nature makes it readable and adjustable by a human[1], it is clearly better suited to be written by an algorithm than by hand. Producing TikZ plot of UPP curves can be in fact a quite tedious and error-prone process.

The `ToTikzPlot()` methods, in fact, address this very problem: they provide an easy interface to obtain TikZ code representing one or more *Nancy* curves, which does away with most of the tedious work, leaving only the fine tuning to the user. Most of the figures provided in this thesis are result of TikZ code written, at least initially, by these methods.

As an example, consider the flow-control service curve $\overline{\beta_{1,2} + 1}$. We can compute the expression and its TikZ plot with the code in Listing 9.1. The resulting TikZ code

Listing 9.1 Example of code to compute and plot a function through TikZ.

```
var f = (new RateLatencyServiceCurve(1, 2) + 1).SubAdditiveClosure();
var tikz = f.ToTikzPlot(settings: new TikzLayoutSettings()
{
    CurveLayout = CurveLayout.SimplifyContinuous,
    GridTickLayout = GridTickLayout.SquareGrid
});
```

is shown in Listing 9.2, and its plot, as rendered by LaTeX, is shown in Figure 9.1.

The library represents, by default, all curves up to to the higher $T_i + 2 \cdot d_i$, with some fallbacks in case of UA curves and similar corner cases, or to the argument `upTo` if the user manually specifies it. Moreover, the library allows to customize how the grid is generated, whether to highlight all breakpoints or to only do so on discontinuities, etc.

[1]If one ignores how easy it is to incur in obscure error messages, that is.

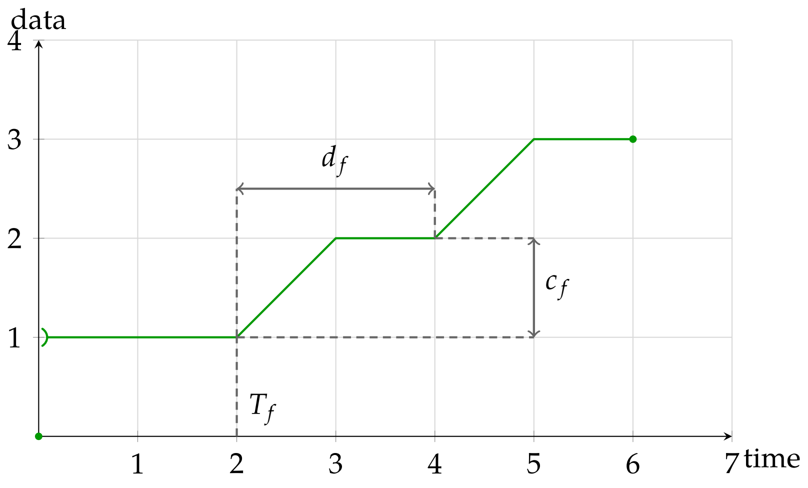


Figure 9.1: The resulting plot.

Listing 9.2 The resulting TikZ code.

```
\begin{tikzpicture}
    \begin{axis}[
        font = \small,
        clip = true,
        grid = major,
        grid style = {draw=gray!30},
        axis lines = left,
        axis equal image,
        xlabel = time,
        ylabel = data,
        x label style = {at={(axis description cs:1,0)},anchor={north west}},
        y label style = {at={(axis description cs:0,1)},rotate=-90,anchor=south},
        xmin = 0,
        ymin = 0,
        xmax = 7,
        ymax = 4,
        xtick = { 1, 2, 3, 4, 5, 6, 7 },
        ytick = { 1, 2, 3, 4 },
        minor xtick = {},
        minor ytick = {}
    ]
        \addplot [ color = green!60!black, thick, only marks, mark size = 1pt ]
            coordinates { (0,0) };
        \addplot [ color = green!60!black, thick, )-, solid, shorten < = 1pt ]
            coordinates { (0, 1) (2, 1) (3, 2) (4, 2) (5, 3) (6, 3) };
        \addplot [ color = green!60!black, thick, only marks, mark size = 1pt ]
            coordinates { (6,3) };
        \addplot [ color = black!60, thick, densely dashed ]
            coordinates { (2, 0) (2, 1) };
        \node [ anchor = south west ] at (axis cs:2, 0) {$T_{f}$};
        \addplot [ color = black!60, thick, densely dashed ]
            coordinates { (4, 2) (4, 2.5) };
        \addplot [ color = black!60, thick, densely dashed ]
            coordinates { (2, 1) (2, 2.5) };
        \addplot [ color = black!60, thick, <-> ]
            coordinates { (2, 2.5) (4, 2.5) };
        \node [ anchor = south ] at (axis cs:3, 2.5) {$d_{f}$};
        \addplot [ color = black!60, thick, densely dashed ]
            coordinates { (2, 1) (5, 1) };
        \addplot [ color = black!60, thick, densely dashed ]
            coordinates { (4, 2) (5, 2) };
        \addplot [ color = black!60, thick, <-> ]
        coordinates { (5, 1) (5, 2) };
        \node [ anchor = west ] at (axis cs:5, 1.5) {$c_{f}$};
    \end{axis}
\end{tikzpicture}
```

Chapter 10

Extending the Algorithmic Toolbox: Lower and Upper Pseudoinverses

The work described in this chapter is part of the contributions in [ZNS23c], coauthored with Paul Nikolaus and Giovanni Stea.

In this chapter, we discuss the extension of the $\mathcal{U}$ algorithmic toolbox to include lower and upper pseudoinverse operators. Henceforth, we will omit the *lower* or *upper* attribute when the discussion applies to both. Moreover, in the follow-up work that is discussed later on in Chapter 14, we introduced a more general version of this operator, i.e., *pseudoinverses over interval*.

First, we recall their formal definitions from Section 4.3.

Definition 4.9 (Lower and Upper Pseudoinverse)**.** Let $f \in \mathcal{U}$ be non-decreasing. Then its *lower pseudoinverse* is defined as

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

and its *upper pseudoinverse* is defined as

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

In DNC, both pseudoinverses are useful to switch from (min,+) to (max,+) algebra and vice versa [Lie17]. They are also used in many results, such as models for round-robin schedulers, as shown in Section 2.4, to compute the horizontal deviation, as shown in Equation (2.3), or as part of the algorithm to compute the composition, as we show in Chapter 11.

The rest of this chapter is organized as follows. In Section 10.1 we show that the pseudoinverse of a function of $\mathcal{U}$ is again in $\mathcal{U}$, and provide expressions to compute its UPP parameters a priori. In Section 10.2 we discuss, first through a visual example and then via pseudocode, how to algorithmically compute the pseudoinverse.

Then, in Section 10.3 we conclude with a summary of the by-curve algorithm, some observations on the algorithmic complexity of this operator. Finally, in Section 10.4 we discuss corner cases.

10.1 Properties of pseudoinverses of functions of $\mathcal{U}$

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* library computes pseudoinverses of generic (non-decreasing) UPP functions, including UC and UI ones.

Lemma 10.1. *Let f be a non-decreasing function of $\mathcal{U}$, then its pseudoinverses $f_\downarrow^{-1}$ and $f_\uparrow^{-1}$ are again in $\mathbb{Q}_+ \to \mathbb{Q} \cup \{+\infty, -\infty\}$ and piecewise affine.*

Proof. Consider first f to be neither UC nor UI. We first observe that for any t where f is linear with slope > 0, it is $f_\downarrow^{-1}(f(t)) = t$ (it behaves as a normal inverse). It also follows from the definition that $f_\downarrow^{-1}(x)$ has a breakpoint in x with value t if f has either a breakpoint in t with value x, or a jump in t from or to x. Since f has breakpoints and limits in $\mathbb{Q}$, it follows that $f_\downarrow^{-1}$ is in $\mathbb{Q}_+ \to \mathbb{Q} \cup \{+\infty, -\infty\}$.

Moreover, as is discussed thoroughly in Section 10.2, $f_\downarrow^{-1}$ is composed of jumps (corresponding to intervals in which f is constant), constant segments (corresponding to breakpoints in which f has a jump) or non-constant segments (corresponding to intervals in which f is non-constant, with inverse slope). Thus, $f_\downarrow^{-1}$ is also piecewise affine.

The case of UC and UI functions is discussed at length in Section 10.4. From that discussion we receive that if f is UC with value x, then $f_\downarrow^{-1}$ is UI from x; and if f is UI from t and last finite value x, then $f_\downarrow^{-1}$ is UC from x with value t. Again, these values are in $\mathbb{Q}$ (due to f).

The same steps can be followed for $f_\uparrow^{-1}$. □

Theorem 10.2 (UPP properties of Lower Pseudoinverse)**.** *Let f be a function of $\mathcal{U}$ that is non-decreasing and neither UC nor UI. Then, its lower pseudoinverse $f_\downarrow^{-1}(x) = \inf\{t \mid f(t) \geq x\}$ is again a function of $\mathcal{U}$ with*

$$T_{f_\downarrow^{-1}} = f\left(T_f + d_f\right), \tag{10.1}$$

$$d_{f_\downarrow^{-1}} = c_f, \tag{10.2}$$

$$c_{f_\downarrow^{-1}} = d_f. \tag{10.3}$$

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), and that $f(t_0) = f(t_1)$.

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,

$$f(\tau - d_f) = f(\tau) - c_f. \tag{10.4}$$

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

$$f(T_f) \leq f(t_1 - d_f) \overset{(10.4)}{=} f(t_1) - c_f < f(t_1) = f(t_0),$$

where we used in the strict inequality that f is not UC. Thus, since $f(t_0) > f(T_f)$, $t_0 > T_f$.

Therefore, for any $k \in \mathbb{N}$,

$$\begin{aligned}
f_\downarrow^{-1}\left(x + k \cdot d_{f_\downarrow^{-1}}\right) &= \inf\left\{t \mid f(t) \geq x + k \cdot d_{f_\downarrow^{-1}}\right\} \\
&\overset{(10.2)}{=} \inf\{t \mid f(t) \geq x + k \cdot c_f\} \\
&= \inf\{t \mid f(t) \geq f(t_1) + k \cdot c_f\} \\
&= \inf\{t \mid f(t) \geq f(t_0) + k \cdot c_f\} \\
&= \inf\{t \mid f(t) \geq f(t_0 + k \cdot d_f)\} \\
&= t_0 + k \cdot d_f \\
&\overset{(10.3)}{=} f_\downarrow^{-1}(x) + k \cdot c_{f_\downarrow^{-1}}.
\end{aligned}$$

Combined with Lemma 10.1, $f_\downarrow^{-1} \in \mathcal{U}$. □

It follows from Theorem 10.2 that, in order to compute a representation $R_{f_\downarrow^{-1}}$, it is sufficient to compute $S^{I'}_{f_\downarrow^{-1}}$ where

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

We use then the following lemma.

Lemma 10.3 (Sufficient Cut for Lower Pseudoinverse). *Let f be a function of $\mathcal{U}$ that is non-decreasing and neither UC nor UI. Then, in order to compute $f_{\downarrow}^{-1}(x)$ with $x \in I' := [0, x'[$, where $x' \geq 0$, it is sufficient to use $f(t)$ with $t \in [0, t']$, where $t' := f_{\downarrow}^{-1}(x')$.*

We provide a proof in Appendix D.3. Using Lemma 10.3, we derive that to compute $S^{I'}_{f_{\downarrow}^{-1}}$ with $I' = \left[0, f(T_f + d_f) + c_f\right[$, it is sufficient to use $S_f^{I_f}$ with

$$I_f = \left[0, T_f + 2 \cdot d_f\right]. \tag{10.5}$$

A similar result can be derived for the upper pseudoinverse.

Theorem 10.4 (UPP properties of Upper Pseudoinverse). *Let f be a function of $\mathcal{U}$ that is non-decreasing and neither UC nor UI. Then, the upper pseudoinverse $f_{\uparrow}^{-1}(x) = \sup\{t \mid f(t) \leq x\}$ is again a function of $\mathcal{U}$ with*

$$T_{f_{\uparrow}^{-1}} = f\left(T_f\right), \tag{10.6}$$

$$d_{f_{\uparrow}^{-1}} = c_f, \tag{10.7}$$

$$c_{f_{\uparrow}^{-1}} = d_f. \tag{10.8}$$

Proof. The proof follows the same steps as the one for the lower pseudoinverse. Let $t_0 \geq T_f$ and $x := f(t_0)$. Moreover, we define

$$t_1 := f_{\uparrow}^{-1}(x) = \sup\{t \mid f(t) \leq x\} = \sup\{t \mid f(t) \leq f(t_0)\}$$

By definition, it is clear that $t_0 \leq t_1$ (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 $t_0 \geq T_f$, it follows that

$$f(t_0 + d_f) \overset{(3.1)}{=} f(t_0) + c_f > f(t_0) = f(t_1),$$

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 $k \in \mathbb{N}$,

$$\begin{aligned}
f_{\uparrow}^{-1}\left(x + k \cdot d_{f_{\uparrow}^{-1}}\right) &= \sup\left\{t \mid f(t) \leq x + k \cdot d_{f_{\uparrow}^{-1}}\right\} \\
&\overset{(10.7)}{=} \sup\{t \mid f(t) \leq x + k \cdot c_f\} \\
&= \sup\{t \mid f(t) \leq f(t_0) + k \cdot c_f\} \\
&= \sup\{t \mid f(t) \leq f(t_1) + k \cdot c_f\} \\
&= \sup\{t \mid f(t) \leq f(t_1 + k \cdot d_f)\} \\
&= t_1 + k \cdot d_f \\
&\overset{(10.8)}{=} f_{\uparrow}^{-1}(x) + k \cdot c_{f_{\uparrow}^{-1}}.
\end{aligned}$$

Combined with Lemma 10.1, $f_{\uparrow}^{-1} \in \mathcal{U}$. □

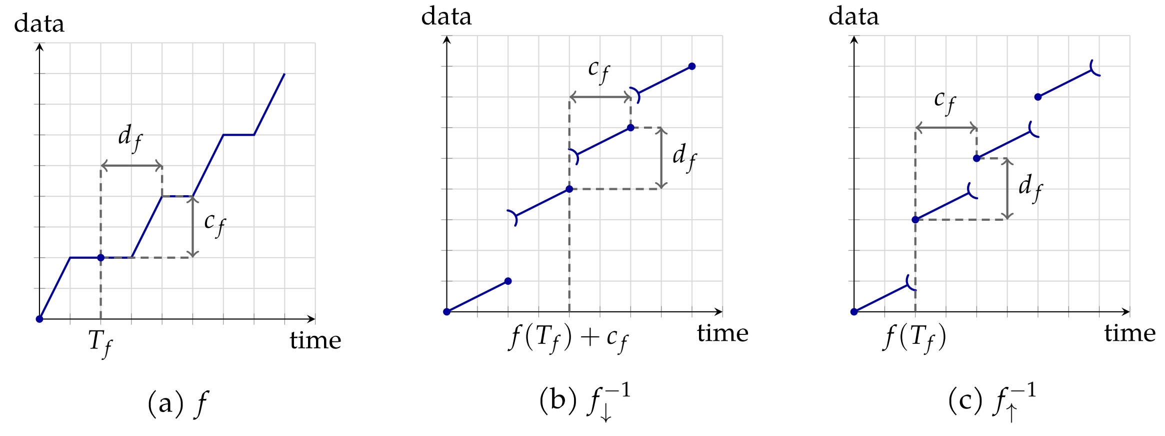


Figure 10.1: Example of f such that $f_\downarrow^{-1}$ is not UPP from $f(T_f)$.

Similar to the previous theorem, it follows from Theorem 10.4 that, in order to compute a representation $R_{f_\uparrow^{-1}}$, it is sufficient to compute $S^{I'}_{f_\uparrow^{-1}}$, where

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

We use then the following lemma.

Lemma 10.5 (Sufficient Cut for Upper Pseudoinverse). *Let f be a function of $\mathcal{U}$ that is non-decreasing and neither UC nor UI. Then, in order to compute $f_\uparrow^{-1}(x)$ with $x \in I' := [0, x'[$, where $x' \geq 0$, it is sufficient to use $f(t)$ with $t \in [0, t']$, where $t' := f_\uparrow^{-1}(x')$.*

We provide a proof in Appendix D.3. Using Lemma 10.5, we derive that to compute $S^{I'}_{f_\uparrow^{-1}}$ with $I' = \left[0, f(T_f) + c_f\right[$, it is sufficient to use $S^{I_f}_f$ with

$$I_f = \left[0, T_f + d_f\right].$$

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 $a < T, b > 0$, as in Figure 10.1. Then $f_\downarrow^{-1}(k) = a$, as the lower pseudoinverse "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 pseudoinverse "goes forward" to the end of the constant segment and $T + b > T$, thus we can rely on the pseudo-periodic property of f.

The above is what can be achieved without information on the *shape* of f. A stronger result can be instead obtained if one goes to the length of checking the values of f at and around T and $T + d$, such as to avoid, e.g., the case described above. This result is discussed as Lemma 14.11 in Chapter 14, as it is a required building block for the results therein.

Moreover, from Lemmas 4.12 and 4.13 we can derive the following corollaries.

Corollary 10.6. *Let $f \in \mathcal{U}$ be non-decreasing and left-continuous. Then, $\left(f_{\uparrow}^{-1}\right)_{\downarrow}^{-1}$ is UPP from T_f, i.e.,*

$$T_{\left(f_{\uparrow}^{-1}\right)_{\downarrow}^{-1}} = T_f.$$

Corollary 10.7. *Let $f \in \mathcal{U}$ be non-decreasing and right-continuous. Then, $\left(f_{\downarrow}^{-1}\right)_{\uparrow}^{-1}$ is UPP from T_f, i.e.,*

$$T_{\left(f_{\downarrow}^{-1}\right)_{\uparrow}^{-1}} = T_f.$$

Note that these provide better lower bounds to the pseudo-period start than applying Theorems 10.2 and 10.4 in sequence.

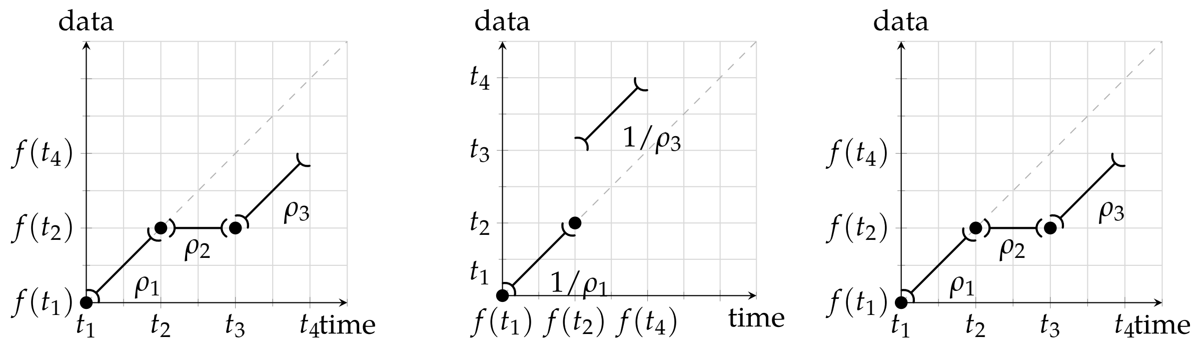
10.2 By-sequence algorithm for pseudoinverses

In this section we discuss the by-sequence algorithms for pseudoinverses. We recall that with "by-sequence" we mean that the operand, and thus its result, has a finite interval as its support. 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 10.2.

We describe Algorithm 2 for the lower pseudoinverse (the one for the upper pseudoinverse differs in few details which we briefly discuss later). Algorithm 2 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 10.1, which require zero, one, two, or three elements to be added to $S_{\downarrow}^{-1}$. These are reported in the



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

Figure 10.2: Example of lower pseudoinverse of a sequence S. Since S is left-continuous, $S = \left(S_{\downarrow}^{-1}\right)_{\downarrow}^{-1}$.

same order in Algorithm 2. The rigorous (though cumbersome) mathematical justification for each case is instead postponed to Appendix D.4 for the benefit of the interested reader.

We exemplify the above algorithm with reference to the example in Figure 10.2. For each of the considered steps, we will reference the case in Table 10.1, the line of Algorithm 2, and the relevant equations from Appendix D.4 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 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 (D.15).

- 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 (D.7).

- 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 (D.11).

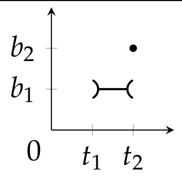
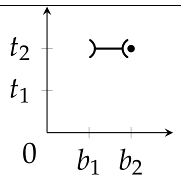
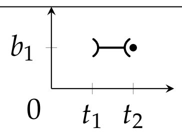
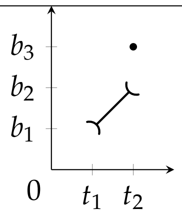
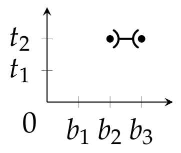
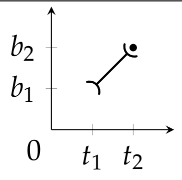
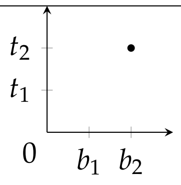
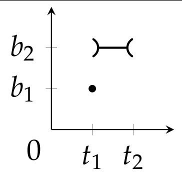
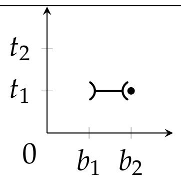
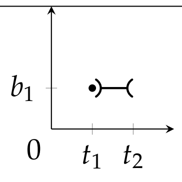
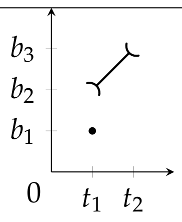
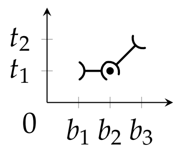
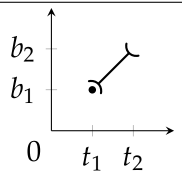
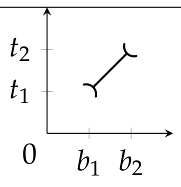
- For the point $p_3 = (t_3, f(t_3))$ with preceding constant segment s_2, we are in case c2, corresponding to Line 10, and no element is added. This follows Equation (D.3).

- 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 (D.15)).

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). As Algorithm 2 performs a linear scan of its input S, its complexity is $\mathcal{O}\left(n(S)\right)$.

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 pseudoinverse 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 10.1: 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 2 Pseudocode for lower pseudoinverse of a finite sequence

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

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

```
1: Define an empty sequence of elements $O := \{\ \}$
2: Append $p := (0, 0)$ to $O$                                   ▷ $f_\downarrow^{-1}(e_0)$
3: for $e_k$ in $(e_2, \dots, e_n)$ do                        ▷ The for loop starts after the origin
4:     if $e_k == p_i$ then                                   ▷ The element is a point
5:         $e_{k-1}$ is a segment $s_{i-1}$
6:         if $s_{i-1}$ is constant then
7:             if $f(t_i^-) < f(t_i)$ then                    ▷ $f$ is not left-cont. at $t_i$
8:                 Append $s := \big(f(t_i^-), f(t_i), t_i, t_i\big)$ to $O$          ▷ (c1)
9:                 Append $p := \big(f(t_i), t_i\big)$ to $O$
10:            else                                           ▷ $f$ is left-cont. at $t_i$
11:                Nothing to append                          ▷ (c2)
12:            end if
13:        else                                               ▷ $s_{i-1}$ is not constant
14:            if $f(t_i^-) < f(t_i)$ then                    ▷ $f$ is not left-cont. at $t_i$
15:                Append $p := \big(f(t_i^-), t_i\big)$ to $O$                 ▷ (c3)
16:                Append $s := \big(f(t_i^-), f(t_i), t_i, t_i\big)$ to $O$
17:                Append $p := \big(f(t_i), t_i\big)$ to $O$
18:            else                                           ▷ $f$ is left-cont. at $t_i$
19:                Append $p := \big(f(t_i), t_i\big)$ to $O$                  ▷ (c4)
20:            end if
21:        end if
22:    else                                                   ▷ The element is a segment $s_i$
23:        $e_{k-1}$ is a point $p_i$
24:        if $e_k = s_i$ is constant then
25:            if $f(t_i) < f(t_i^+)$ then                    ▷ $f$ is not right-cont. at $t_i$
26:                Append $s := \big(f(t_i), f(t_i^+), t_i, t_i\big)$ to $O$          ▷ (c5)
27:                Append $p := \big(f(t_i^+), t_i\big)$ to $O$
28:            else                                           ▷ $f$ is right-cont. at $t_i$
29:                Nothing to append                          ▷ (c6)
30:            end if
31:        else                                               ▷ $s_i$ is not constant
32:            if $f(t_i) < f(t_i^+)$ then                    ▷ $f$ is not right-cont. at $t_i$
33:                Append $s := \big(f(t_i), f(t_i^+), t_i, t_i\big)$ to $O$          ▷ (c7)
34:                Append $p := \big(f(t_i^+), t_i\big)$ to $O$
35:                Append $s := \big(f(t_i^+), f(t_{i+1}^-), t_i, t_{i+1}\big)$ to $O$
36:            else                                           ▷ $f$ is right-cont. at $t_i$
37:                Append $s := \big(f(t_i^+), f(t_{i+1}^-), t_i, t_{i+1}\big)$ to $O$   ▷ (c8)
38:            end if
39:        end if
40:    end if
41: end for
```

10.3 By-curve algorithm for pseudoinverses

We can now discuss the by-curve algorithm by combining the results presented in Sections 10.1 and 10.2. In Algorithm 3, we show the pseudocode to compute $f_{\downarrow}^{-1}$ for a function f of $\mathcal{U}$. The analogous for upper pseudoinverse, which we omit for brevity, can be similarly derived from the results in the sections above.

Algorithm 3 Pseudocode for lower pseudoinverse of a function of $\mathcal{U}$

Input Representation R_f of a non-decreasing function f of $\mathcal{U}$, consisting of sequence S_f and parameters T_f, d_f and c_f.

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

```
1: Compute the UPP parameters for the result                    ▷ Theorem 10.2
2:     $T_{f_{\downarrow}^{-1}} \leftarrow f\left(T_f + d_f\right)$
3:     $d_{f_{\downarrow}^{-1}} \leftarrow c_f$
4:     $c_{f_{\downarrow}^{-1}} \leftarrow d_f$
5: Compute $S_f^I$                                              ▷ Equation (10.5)
6:     $D \leftarrow \left[0, T_f + 2 \cdot d_f\right[$
7:     $S_f^I \leftarrow \mathrm{Cut}(R_f, D)$
8: Compute $S_{f_{\downarrow}^{-1}} \leftarrow \left(S_f^I\right)_{\downarrow}^{-1}$          ▷ Algorithm 2
9: $R_{f_{\downarrow}^{-1}} \leftarrow \left(S_{f_{\downarrow}^{-1}}, T_{f_{\downarrow}^{-1}}, d_{f_{\downarrow}^{-1}}, c_{f_{\downarrow}^{-1}}\right)$
```

Regarding the complexity of Algorithm 3, we note that the main cost is computing $\left(S_f^I\right)_{\downarrow}^{-1}$. Since Algorithm 2 is a linear scan of the input sequence, the resulting complexity is $\mathcal{O}\left(n\left(S_f^I\right)\right)$.

10.4 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 $t \geq T_f$.[1] 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 pseudoinverses. We provide formal definitions below.

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

Definition 10.8. Let $f \in \mathcal{U}$ be UC, and let $C := \lim_{t \to +\infty} f(t)$, $C \in \mathbb{Q}$, be its (ultimately) constant value. Then, we define

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

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 10.9. 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 10.8 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 10.9 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 3 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 10.10. *Let $f \in \mathcal{U}$ be a non-decreasing, UC function with $T_C \in \mathbb{Q}_+$. If $f(T_C) < C$, its lower pseudoinverse $f_\downarrow^{-1}(y)$ 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 pseudoinverse $f_\uparrow^{-1}(y)$ 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 pseudoinverse $f_\downarrow^{-1}(y)$ 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 pseudoinverse $f_\uparrow^{-1}(y)$ 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 pseudoinverses are UI with $T_I = C$.

Proposition 10.11. *Let $f \in \mathcal{U}$ be a non-decreasing, UI function with $T_I \in \mathbb{Q}_+$. Then, its lower pseudoinverse $f_\downarrow^{-1}(y)$ 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 pseudoinverse $f_\uparrow^{-1}(y)$ 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 pseudoinverses are UC with $T_C = L$.[2]

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

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.

Chapter 11

Extending the Algorithmic Toolbox: Composition

The work described in this chapter is part of the contributions in [ZNS23c], coauthored with Paul Nikolaus and Giovanni Stea.

In this chapter, we discuss the extension of the $\mathcal{U}$ algorithmic toolbox to include the composition operator, i.e., $(f \circ g)(t) = f(g(t))$.

This chapter is organized as follows. In Section 11.1 we show that the composition of functions in $\mathcal{U}$ is again a function of $\mathcal{U}$, and provide expressions to compute its UPP parameters a priori. In Section 11.2 we discuss, first via an example and then via pseudocode, how to compute the composition algorithmically. Finally, in Section 11.3 we conclude with a summary of the by-curve algorithm and some observations on the algorithmic complexity of this operator.

11.1 Properties of composition of functions of $\mathcal{U}$

We assume that the inner function g is not UI.[1] 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 11.1 (UPP properties of Composition). *Let f and g be two functions of $\mathcal{U}$ with g being non-negative, non-decreasing and not UI. Then, their composition $h := f \circ g$ is again a function of $\mathcal{U}$ with*

$$T_h = \max\left(g_{\downarrow}^{-1}(T_f), T_g\right), \tag{11.1}$$

$$d_h = p_{d_f} \cdot d_g \cdot q_{c_g}, \tag{11.2}$$

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

$$c_h = q_{d_f} \cdot p_{c_g} \cdot c_f, \tag{11.3}$$

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. It is trivial to see that, being f and g piecewise affine $\mathbb{Q}_+ \to \mathbb{Q} \cup \{+\infty, -\infty\}$ functions, their composition $f \circ g$ is again piecewise affine $\mathbb{Q}_+ \to \mathbb{Q} \cup \{+\infty, -\infty\}$. This point is further stressed in Section 11.2, were we describe an algorithm to get the points and segments composing $f \circ g$.

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

$$\begin{aligned} h(t + k_h \cdot d_h) &= f\left(g(t + k_h \cdot d_h)\right) \\ &= f\left(g\left(t + k_h \cdot \frac{d_h}{d_g} \cdot d_g\right)\right) \\ &\overset{(3.1)}{=} f\left(g(t) + k_h \cdot \frac{d_h}{d_g} \cdot c_g\right), \end{aligned}$$

where we used the UPP property of g in the last line. Note that $k_g := k_h \cdot \frac{d_h}{d_g} \in \mathbb{N}$, since $\frac{d_h}{d_g} \overset{(11.2)}{=} p_{d_f} \cdot q_{c_g} \in \mathbb{N}$, 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

$$\begin{aligned} h(t + k_h \cdot d_h) &= f\left(g(t) + k_h \cdot \frac{d_h}{d_g} \cdot c_g\right) \\ &= f\left(g(t) + k_h \cdot \frac{d_h \cdot c_g}{d_g \cdot d_f} \cdot d_f\right) \\ &\overset{(3.1)}{=} f(g(t)) + k_h \cdot \frac{d_h \cdot c_g}{d_g \cdot d_f} \cdot c_f \\ &= h(t) + k_h \cdot \frac{d_h \cdot c_g \cdot c_f}{d_g \cdot d_f} \\ &\overset{(11.3)}{=} h(t) + k_h \cdot c_h. \end{aligned}$$

Note that $k_f := k_h \cdot \frac{d_h \cdot c_g}{d_g \cdot d_f} \in \mathbb{N}_0$, since $\frac{d_h \cdot c_g}{d_g \cdot d_f} \overset{(11.2)}{=} \frac{p_{d_f}}{d_f} \cdot q_{c_g} \cdot c_g = q_{d_f} \cdot p_{c_g} \in \mathbb{N}_0$ 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 pseudoinverse and g being non-decreasing, the latter expression implies that $t \geq g_{\downarrow}^{-1}(T_f)$ (Equation (4.9)). Therefore, we require

$$t \geq T_h \overset{(11.1)}{=} \max\left(g_{\downarrow}^{-1}(T_f), T_g\right).$$

This concludes the proof. □

Remark 11.2. 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

$$d_h \overset{(11.2)}{=} p_{d_f} \cdot q_{c_g} \cdot d_g = d_f \cdot d_g,$$

and the properties are then verified since $\frac{d_h}{d_g} = d_f \in \mathbb{N}$ 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 11.1 that to compute a representation R_h, it is sufficient to compute $S_h^{I_h}$, where

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

It follows that

$$S_h^{I_h} = S_f^{I_f} \circ S_g^{I_g},$$

where

$$\begin{aligned} I_g &= [0, T_h + d_h[, \\ I_f &= \left[g(0), g\left((T_h + d_h)^-\right)\right]. \end{aligned} \tag{11.4}$$

The reason I_f needs to be right-closed is that $S_g^{I_g}$ may end with a constant segment. If this happens, $\exists \bar{t} \in I_g$ 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 I_f. On the other hand, if $S_g^{I_g}$ ends with a strictly increasing segment, it is safe to have I_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 11.3. *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*

$$\begin{aligned} T_h &= \max\left(g_{\downarrow}^{-1}(T_f), T_g\right), \\ d_h &= \frac{d_f}{\rho_g}, & (11.5) \\ c_h &= c_f. & (11.6) \end{aligned}$$

We provide a proof in Appendix E.1. To compute the R_h it is sufficient to use

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

It follows that

$$S_h^{I_h} = S_f^{I_f} \circ S_g^{I_g},$$

where

$$\begin{aligned}
I_g &= [0, T_h + d_h[\\
&\overset{(11.5)}{=} \left[0, T_h + \frac{d_f}{\rho_g}\right[, \\
I_f &= \left[g(0), g\left((T_h + d_h)^-\right)\right[\\
&\overset{(11.5)}{=} \left[g(0), g\left(T_h + \frac{d_f}{\rho_g}\right)\right[\\
&= [g(0), g(T_h) + d_f[.
\end{aligned} \tag{11.7}$$

Here, we observe that interval I_f is smaller than the one obtained by applying directly Theorem 11.1, due to the disappearance of a factor $q_{d_f} \cdot p_{c_g} \geq 1$. In fact, with Theorem 11.1 we would have:

$$\begin{aligned}
I_g &= [0, T_h + d_h[\\
&\overset{(11.2)}{=} \left[0, T_h + p_{d_f} \cdot d_g \cdot q_{c_g}\right[\\
&= \left[0, T_h + q_{d_f} \cdot p_{c_g} \cdot \frac{d_f}{\rho_g}\right[, \\
I_f &= \left[g(0), g\left((T_h + d_h)^-\right)\right[\\
&\overset{(11.2)}{=} \left[g(0), g\left(\left(T_h + p_{d_f} \cdot d_g \cdot q_{c_g}\right)^-\right)\right[\\
&= \left[g(0), g(T_h) + q_{d_f} \cdot p_{c_g} \cdot d_f\right[.
\end{aligned}$$

As specified in the statement of Proposition 11.3, we exclude the case when g is UC. This is because of Equation (11.5) 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 E.2. Next, we consider the case when only f is UA.

Proposition 11.4. *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*

$$\begin{aligned}
T_h &= \max\left(g_\downarrow^{-1}(T_f), T_g\right), \\
d_h &= d_g, \qquad (11.8) \\
c_h &= c_g \cdot \rho_f. \qquad (11.9)
\end{aligned}$$

We provide a proof in Appendix E.1. To compute R_h it is sufficient to use

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

It follows that

$$S_h^{I_h} = S_f^{I_f} \circ S_g^{I_g},$$

where

$$\begin{aligned} I_g &= [0, T_h + d_h[\\ &\overset{(11.8)}{=} \left[0, T_h + d_g\right[, \\ I_f &= \left[g(0), g\left((T_h + d_h)^-\right)\right] \\ &\overset{(11.8)}{=} \left[g(0), g\left((T_h + d_g)^-\right)\right]. \end{aligned} \tag{11.10}$$

Again, interval I_g is smaller than the one that Theorem 11.1 would yield, due to the disappearance of a factor $p_{d_f} \cdot q_{c_g} \geq 1$. For comparison, Theorem 11.1 yields

$$\begin{aligned} I_g &= [0, T_h + d_h[\\ &\overset{(11.2)}{=} \left[0, T_h + p_{d_f} \cdot q_{c_g} \cdot d_g\right[, \\ I_f &= \left[g(0), g\left((T_h + d_h)^-\right)\right] \\ &\overset{(11.2)}{=} \left[g(0), g\left(\left(T_h + p_{d_f} \cdot d_g \cdot q_{c_g}\right)^-\right)\right]. \end{aligned}$$

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

Proposition 11.5. *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*

$$\begin{aligned} T_h^a &= \max\left(g_{\downarrow}^{-1}(T_f^a), T_g^a\right), \\ \rho_h &= \rho_f \cdot \rho_g. \end{aligned} \tag{11.11}$$

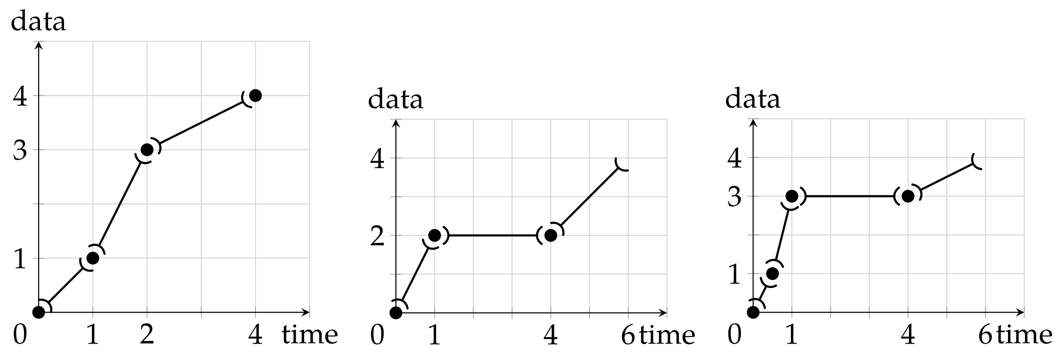
We provide a proof in Appendix E.1. Considering Equation (11.4), we observe how taking these results into account will yield tighter I_f, I_g than what we obtain with Theorem 11.1.

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 E.2.

11.2 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 $[g(0), g(t^-)]$.[2] Then, $S_h = S_f \circ S_g$ is the sequence representing

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



(a) S_f (b) S_g (c) $S_f \circ S_g$

Figure 11.1: Example of composition of two sequences.

$h = f \circ g$ over the interval $[0, t[$. We use the example shown in Figure 11.1, 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

$$\begin{aligned} &g(x) = g(a^+) + \rho_g \cdot (x - a), \qquad \forall x \in \,]a, b[\,, \\ &f(x) = f\left(g(a^+)^+\right) + \rho_f \cdot \left(x - g(a^+)\right), \qquad \forall x \in \left]g(a^+), g(b^-)\right[\,, \end{aligned} \tag{11.12}$$

where we use the shorthand notation

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

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

More broadly speaking, we have a segment of g mapping to a segment of f. In the example of Figure 11.1, $]4, 6[$ 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 (11.12) 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 (11.12) 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 pseudoinverse of g to find the corresponding $\bar{t}_i = g_\downarrow^{-1}(y_i)$.[3] 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 11.1, $]0, 4[$ is such an interval:

[3]Following the discussion in Section 4.3, $\left(S_g\right)_\downarrow^{-1}$ is sufficient for this computation.

- for S_g we find the set $\{t_1 = 1\}$;
- for S_f we find the set $\{y_1 = 1\} \rightarrow \left\{\bar{t}_1 = \frac{1}{2}\right\}$;
- the combined set of breakpoints is then $\left\{\frac{1}{2}, 1\right\}$, and the open intervals that verify Equation (11.12) is $\left\{\left]0, \frac{1}{2}\right[, \left]\frac{1}{2}, 1\right[,]1, 4[\right\}$.

By generalizing this reasoning, we obtain Algorithm 4.

Algorithm 4 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 $O = \{o_1, \dots, o_m\}$.
1: Define an empty sequence of elements $O := \{\,\}$
2: Let $\mathbb{T}$ be an empty, but ordered set of distinct rationals
3: Let $P_g$ be the set of points of $S_g$
4: for $p_i$ in $P_g$ do
5:     Add the time $t_i$ of $p_i$ to $\mathbb{T}$
6: end for
7: Let $P_f$ be the set of points of $S_f$, excluding the last point $g(a^-)$
8: for $p_i$ in $P_f$ do
9:     Given time $t_i$ of $p_i$, add $\bar{t}_i = g_\downarrow^{-1}(t_i)$ to $\mathbb{T}$      ▷ preserving the order in $\mathbb{T}$
10: end for
11: for each pair of consecutive $(t_i, t_{i+1})$ in $\mathbb{T}$ do
12:     Append $p := (t_i, f(g(t_i)))$ to $O$
13:     Append $s := \left(t_i, t_{i+1}, f\left(g(t_i^+)^+\right), f\left(g(t_{i+1}^-)^-\right)\right)$ to $O$
14: end for
```

11.3 By-curve algorithm for composition

We can now discuss the by-curve algorithm, by combining the results presented in Sections 11.1 and 11.2. In Algorithm 5 we show the pseudocode to compute the composition $h = f \circ g$ of functions f and g of $\mathcal{U}$, 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 interval computations.

Regarding the complexity of Algorithm 5, we note that the main cost is computing $S_h \leftarrow S_f^{I_f} \circ S_g^{I_g}$. Note that Algorithm 4 can be implemented as a linear scan of $S_f^{I_f}$ and $S_g^{I_g}$. In fact, even though sampling a function or its pseudoinverse has, in

Algorithm 5 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 $h = f \circ g$.

```
1:  Compute the UPP parameters for the result                ▷ Theorem 11.1
2:      $T_h \leftarrow \max\left(g_\downarrow^{-1}(T_f), T_g\right)$
3:      $d_h \leftarrow p_{d_f} \cdot d_g \cdot q_{c_g}$
4:      $c_h \leftarrow q_{d_f} \cdot p_{c_g} \cdot c_f$
5:  Compute $S_f^{I_f}$ and $S_g^{I_g}$                      ▷ Equation (11.4)
6:      $I_f \leftarrow [g(0), g((T_h + d_h)^-)[$
7:      $S_f^{I_f} \leftarrow \mathrm{Cut}(R_f, I_f)$
8:      $I_g \leftarrow [0, T_h + d_h[$
9:      $S_g^{I_g} \leftarrow \mathrm{Cut}(R_g, I_g)$
10: Compute $S_h \leftarrow S_f^{I_f} \circ S_g^{I_g}$       ▷ Algorithm 4
11: $R_h \leftarrow (S_h, T_h, d_h, c_h)$
```

general, logarithmic cost (see Section 9.1), in Algorithm 4 these queries can be done *in increasing order*, thus it can be implemented as an iteration over the values of the two sequences. The resulting complexity is therefore $\mathcal{O}\left(n\left(S_f^{I_f}\right) + n\left(S_g^{I_g}\right)\right)$.

Note that given the expressions in Theorem 11.1, 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 11.3 to 11.5 yields performance improvements, since I_f and I_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 10.1).

11.4 Example study on IWRR scheduler

In this section, we recall the discussion in Section 2.4 and Section 6.2, where we discussed examples of literature result that require multiple computation steps involving operators such as pseudoinverse and composition. In particular, we use the example provided by [TLB21, Theorem 1], which uses the composition operator to compute the per flow strict service curve for an Interleaved Weighted Round Robin scheduler.

We recall the contents of this theorem below, though for the purpose of this proof of concept, we focus on the steps of the computations which will then translate into code. Computing this service curve for a 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 $\beta_i = \gamma_i \circ \beta$.[4] We restate here Theorem 2.7, while its code translation using *Nancy* is in Listing 11.1. Figure 11.2 shows the resulting plot, obtained running the code with .NET Notebook. It should be easy for the reader to check that the code maps easily to the mathematical expressions, attesting to the design goals we aimed for in Chapter 8.

Theorem 2.7 (Strict Per-Flow Service Curves for IWRR). *Assume n flows arriving at a server performing Interleaved Weighted Round Robin with weights $w_1, \dots, w_n$. Let $l_i^{\min}$ and $l_j^{\max}$ denote the minimum and maximum packet size of the respective flow. Let this server offer a superadditive 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

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

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

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

In the example in [TLB21, Figure 3], β is a constant-rate service curve, thus UA, while γ_i is, in general, a UPP function. On 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 11.1 and its specialized version for UA inner functions, Proposition 11.3, and

[4]Recall that composition requires the lower pseudoinverse of the inner function to be computed, hence this example makes use of both the algorithms presented in this thesis.

Listing 11.1 Code used to replicate the results of [TLB21, 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);
```

Table 11.1: 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

that we can expect the latter to be more efficient due to the tighter I_f, as explained below Equation (11.7).

Our experiments confirm the above intuition. We run the computation using the same parameters of the example in [TLB21, Figure 3], on System 4. As shown in Table 11.1, when using Theorem 11.1, computing the result took a median of 1.11 seconds. On the other hand, using Proposition 11.3 the same result is obtained in 0.55 milliseconds in the median, an improvement of three orders of magnitude.

Since the output of this model is a UPP curve, on needs support for UPP convolutions to use it in studies that involve other nontrivial models. We exemplify this with study involving two IWRR schedulers in tandem.[5] We use a function that

[5]The same example is available as a tutorial on the documentation website [ZSb] and as .NET

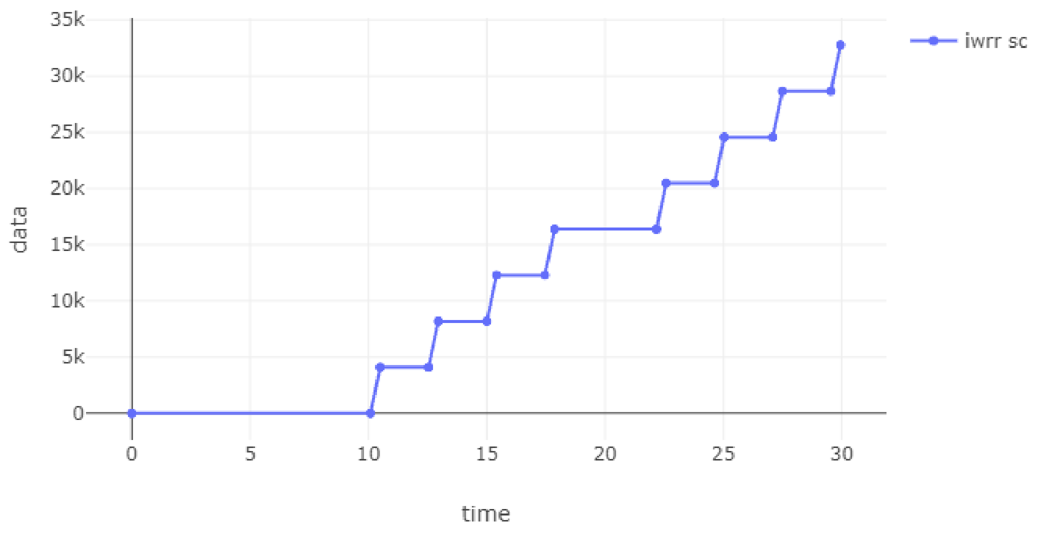


Figure 11.2: Plot of the resulting service curve β_i.

Listing 11.2 Signature of the function generalizing the IWRR service curve computation.

```
Curve IwrrServiceCurve(
    int foi, int[] weights, int[] l_min, int[] l_max, SuperAdditiveCurve beta
)
```

generalizes Listing 11.1, with the signature in Listing 11.2.

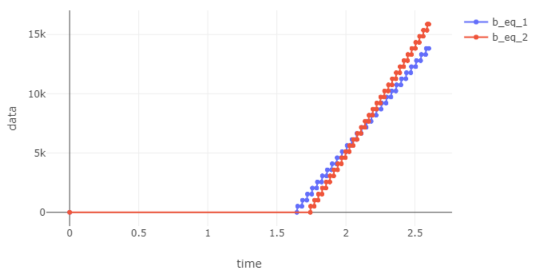
Thus, once we specify the parameters of our example, we can compute the service curves for the schedulers and derive and equivalent service curve for the tandem – using which we can upper bound the worst-case delay and backlog, as we show in Listing 11.3. We show plots of the curve obtained in Figure 11.3, obtained running the code with .NET Notebook.

The convolution shown in this example involves two generic UPP curve which, as we mentioned, can become quite computationally costly depending on the numerical properties of the inputs. Moreover, these curves are not subadditive, so the optimizations discussed in Chapter 13 cannot be applied. They can benefit, however, from the optimizations discussed in Chapter 14: we will reconsider this example there, and discuss the performance improvement.

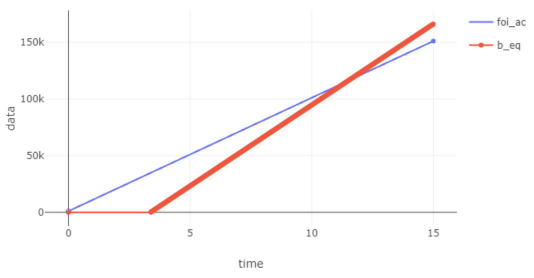
Notebook on Github [ZSd].

Listing 11.3 Example study of a tandem of IWRR schedulers.

```
// ms as time unit, bit as data unit
// the flow has a max arrival rate of 10 Mbps, and is guaranteed such bandwidth at each node
var foi_ac = new SigmaRhoArrivalCurve(1024, 10000);

// 100 Mb/s, 1 ms of latency
// 3 cross flows
var b1 = new RateLatencyServiceCurve(100000, 1);
var weights_1 = new []{10, 30, 40, 20};
var l_min_1 = new []{512, 512, 512, 512};
var l_max_1 = new []{1024, 1024, 1024, 1024};

// 200 Mb/s, 1 ms of latency
// 5 cross flows
var b2 = new RateLatencyServiceCurve(200000, 1);
var weights_2 = new []{10, 20, 40, 50, 30, 50};
var l_min_2 = new []{512, 512, 512, 512, 512, 512};
var l_max_2 = new []{1024, 1024, 1024, 1024, 1024, 1024};

var b_eq_1 = IwrrServiceCurve(0, weights_1, l_min_1, l_max_1, b1);
var b_eq_2 = IwrrServiceCurve(0, weights_2, l_min_2, l_max_2, b2);

var b_eq = Curve.Convolution(b_eq_1, b_eq_2);
var delay_bound = Curve.HorizontalDeviation(foi_ac, b_eq);
// Output: 3,46 ms
var backlog_bound = Curve.VerticalDeviation(foi_ac, b_eq);
// Output: 34900 bit
```



(a) Service curves computed for the IWRR schedulers.



(b) Tandem service curve vs. flow of interest arrival curve.

Figure 11.3: Plots of the curves, obtained running the code with .NET Notebook

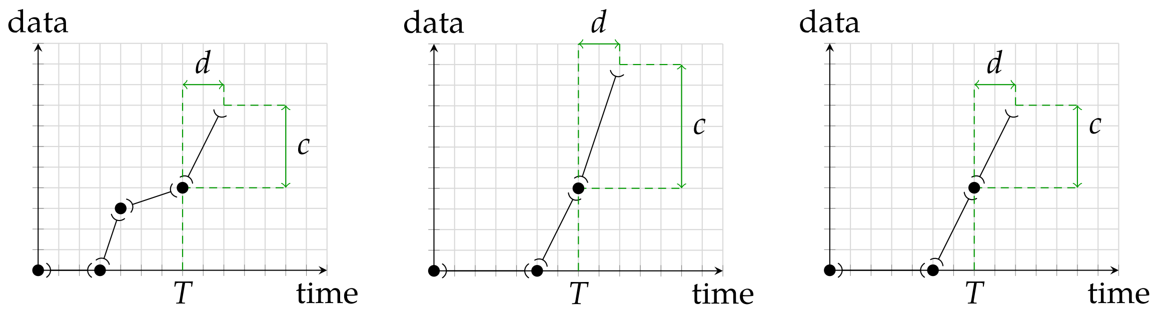
Chapter 12

Representation Minimization

The work described in this chapter is part of the contributions in [ZS23]*, coauthored with Giovanni Stea. We wish to thank Paul Nikolaus for the help on improving the formalization of this work.*

In this chapter, we present *representation minimization*, i.e., a set of algorithms that seeks to minimize the complexity and memory occupation of a `Curve` object by minimizing the representation of its function f. As mentioned, given a representation $R_f = (S, T, d, c)$ of a function f of $\mathcal{U}$, its cardinality $n(S)$ and parameters d and T are the main factors for the algorithmic complexity of operations involving it. We can see this by observing that $n(S)$ is directly correlated with the memory occupied by the `Sequence` object, while, as discussed before, the parameters T and d are used to compute values, such as the cut intervals, which are positively correlated with the performance cost of operations. A first way to abate computation times is therefore to find the *minimal* representation of f.
We say that two representations R_f and R_g are *equivalent* if they represent the same function, i.e., $f(t) = g(t)\ \forall t \geq 0$. A *minimal* representation $\tilde{R}$ is such that, given any equivalent representation R, then $n(\tilde{S}) \leq n(S)$. We note that more than one minimal representation may exist, as in some cases we can have representations with the same minimal $n(\tilde{S})$ but different d and T. For the cases for which this is *attainable*, it is desirable to have the minimal representation that minimizes also d and T. As we will show, our algorithm does exactly this: it yields a minimal representation and, if that is attainable, this will also be the representation with minimal d and T.

Unfortunately, the generic algorithms for operations on functions of $\mathcal{U}$ (such as the ones described in [BT08; BBL18] and recalled in Chapter 3, or the ones we introduce in Chapters 10 and 11) do not yield minimal representations, even when the representations of their operands are minimal. The steps described in Chapter 3, in fact, compute the smallest values that can be formally proved to be valid *a priori*, with no knowledge of the shape of the result. These values can be much larger than



(a) First operand f. (b) Second operand g. (c) Result of $f \wedge g$.

Figure 12.1: Example of non-minimal result of a minimum operation.

those of a minimal representation.

A simple example is given in Figure 12.1. Starting from the parameters of the operands f and g, the algorithm computes $T = 7$ for the result $f \wedge g$. However, we can see that the result is actually a rate-latency curve that can be described with $T = 5$. This phenomenon – that we have just exemplified using a minimum operation – affects convolution as well, and it is especially impactful when many convolutions are required, such as in a SAC or in (2.9), where the result of one is in fact the operand of the next. In fact, we recall that the cost of the convolution is superquadratic with the size of the extended representations of the operands (Chapter 3 and section 8.8).

Note that there is no efficient way – that we know of – to predict the minimum representation *a priori*, i.e., before the operation is computed. This is basically because the result depends on unpredictable numerical properties of the operands (e.g., the segment endpoints). We therefore introduce an algorithm to minimize the representation *a posteriori*, i.e., after the result of the operation has been computed. We will show later that minimization is computationally cheap, and may yield considerable speedups.

We note that [BT07, p. 41] first observed this problem, showing that the *compressed form* of a function (which should be the same as what we call minimal representation, although this is not formally defined therein) can be computed in linear time. With respect to that work, we provide a sound definition of minimal representation, discuss if and when it is attainable, we devise algorithms to obtain it, and we investigate the impact of representation minimization on the efficiency of NC computations.

Definition 3.7 (Breakpoint). Let f be a function of $\mathcal{U}$. We say that t_b is a *breakpoint* of f if f is non-differentiable in t_b, i.e., either of the following is true:

- f has a discontinuity at t_b;

- the rates of f in t_b^- and t_b^+ are different.

A first thing to do is to ensure that the sequence in a representation is *well-formed*. We say that S is a *well-formed* sequence if the abscissa of any point in S is a breakpoint of f. In other words, in a well-formed sequence there are no *unnecessary* points.[1]

As we anticipated, the generic algorithms for minimum and convolution may not yield well-formed sequences, even when the sequences of their operands are well-formed. However, recovering well-formedness only takes a simple $\mathcal{O}\left(n(S)\right)$ check of the resulting sequence S, to find segment-point-segment triplets that can be merged, i.e., replaced with a single segment. This is done on-the-fly in *Nancy*, using the *generator* method `IEnumerable<Element>.Merge()`. From now on, we will therefore assume that sequences are well-formed, without loss of generality. We describe below a minimization algorithm consisting of two phases:

- minimization of the period;
- minimization of the transient.

Hereafter, we denote with $S^{\mathcal{T}}$ the *transient part* of a sequence S (i.e., in interval $\mathcal{T} = [0, T[$) and with $S^{\mathcal{P}}$ its *periodic part* (i.e., in interval $\mathcal{P} = [T, T+d[$).

12.1 Minimization of the period

We set to finding the minimal period $\tilde{d}$, defined as follows:

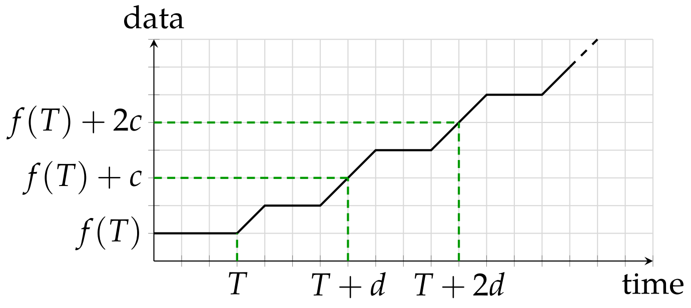
Definition 12.1. A minimal period $\tilde{d}$ for f is such that $R_f = (S, T, \tilde{d}, \tilde{c})$ is a representation of f, and there exists no $q \in (0,1) \subset \mathbb{Q}$ such that

$$f(t + q \cdot \tilde{d}) = f(t) + q \cdot \tilde{c} \qquad \text{for all } t \geq T.$$

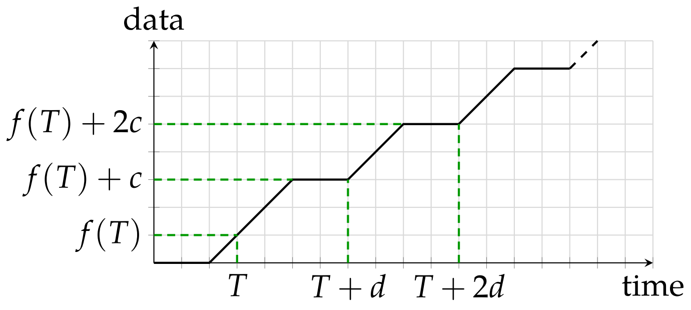
Period minimization is only relevant if the function is not *Ultimately Affine* (UA). Recalling its definition (Definition 3.3), a UA function ends, graphically, with a half-line. This is equivalent to saying that f has no breakpoint for any $t > T$ and, conversely, any $f \in \mathcal{U}$ having a breakpoint $t_b > T$ is not UA.

Note that it is important that inequality $t > T$ is strict. In fact, whether T itself is a breakpoint or not may depend on the transient behavior. On the other hand, we are interested in the periodic behavior. Therefore, we check if $T + d$ is a breakpoint, i.e., if point $(T, f(T))$, repeated after a period in $(T + d, f(T) + c)$, breaks the linear behavior between one pseudo-period and the next. Figure 12.2 shows two examples to illustrate the above. For this reason, in the following we will focus on the breakpoints in $]T, T + d]$.

[1] An exception must be made at T, where a point has to be inserted regardless, marking the end of the transient and the beginning of the periodic part, because this simplifies the implementation. Such an exception has no impact on the rest of our discussion.



(a) Example of function f with a breakpoint in T, but not in $T + k \cdot d$.



(b) Example of function without a breakpoint in T, but with breakpoints in $T + k \cdot d$.

Figure 12.2: Breakpoints in T vs. in $T + d$.

A UA function has no minimal period. In fact, its period has an arbitrary length $d > 0$. Accordingly, its $S^{\mathcal{P}}$ only consists of point $(T, f(T))$ and a segment of length d and slope ρ_f, hence $c = \rho_f \cdot d$. For UA functions, then, there is just nothing to do. Conversely, any f which is not UA has a minimal period. In fact, call t_b the *leftmost* breakpoint such that $t_b > T$ (we know that there is at least one): then, the interval $]T, T + d]$ must include t_b, since it must include at least one breakpoint if f is not UA. Then, $d \geq t_b - T > 0$.

The next question, then, is what characterizes non-minimal periods and how we can find the minimal one, given a representation. The first step is recognizing that non-minimal periods are integer multiples of the minimal one. We provide both visual examples – such as Figure 12.3, discussed later on – and a formal proof.

Proposition 12.2. *Let $f \in \mathcal{U}$ be a non-UA function, and let $\tilde{d}$ be its minimal period. Then, for any period d, it holds that*

$$d/\tilde{d} \in \mathbb{N}.$$

Proof. We define the integer part of $d/\tilde{d}$ as $k := \lfloor d/\tilde{d} \rfloor$ and the fractional part of $d/\tilde{d}$ as $q := d/\tilde{d} - \lfloor d/\tilde{d} \rfloor \in [0, 1[$, thus $d/\tilde{d} = k + q$. Then, it holds that

$$\begin{aligned} f(t + d) &= f(t + (k + q)\tilde{d}) \\ &= f(t + q\tilde{d} + k \cdot \tilde{d}) \end{aligned}$$

$$
\begin{aligned}
&\overset{(3.1)}{=} f(t + q\tilde{d} + (k-1)\cdot\tilde{d}) + c \\
&\overset{(3.1)}{=} \dots \\
&\overset{(3.1)}{=} f(t + q\tilde{d}) + k \cdot c.
\end{aligned}
$$

As the period $\tilde{d}$ is assumed to be minimal, if $0 < q < 1$ then we observe $f(t + q\tilde{d}) = f(t) + q\tilde{c}$ cannot apply, thus the UPP property does not hold for d. Thus, d cannot be an equivalent representation unless $q = 0$. □

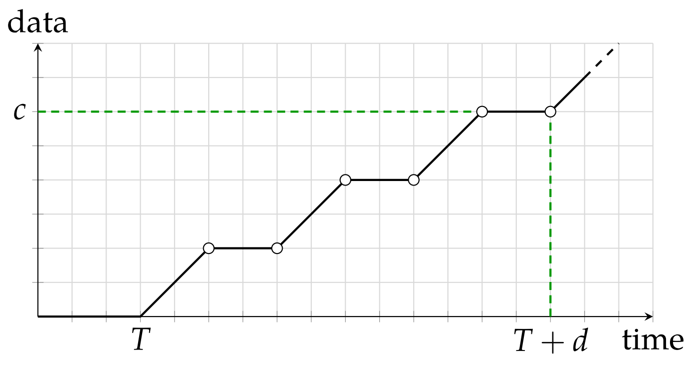
Then, given $R_f = (S, T, d, c)$, if f also admits a minimal period d', it must hold that $d/d' = c/c' = p$, where $p \in \mathbb{N}$. Such p also divides $S^{\mathcal{P}}$ in p *matching parts*, i.e., such that $\forall t \in \left]T, T + \frac{d}{p}\right]$ and $k \in \mathbb{N}$ it holds that $f(t + k \cdot \frac{d}{p}) = f(t) + k \cdot \frac{c}{p}$. Hence, we call p a *divisor of* $S^{\mathcal{P}}$. We exemplify this in Figure 12.3. Figure 12.3a shows f and its representation, with breakpoints in $]T, T + d]$ highlighted as circles. Figure 12.3b shows that $d/3$ is also a (minimal) period, and that – accordingly – $S^{\mathcal{P}}$ consists of $p = 3$ consecutive replicas of a smaller periodic part $S^{\mathcal{P}'}$, which is highlighted in red in the figure. Thus, in order to minimize the period of a non-UA function, we need to find the possible divisors of $S^{\mathcal{P}}$, i.e., to test if the latter is in fact the juxtaposition of matching parts. This can be done efficiently by observing the following.

Lemma 12.3. *Let $f \in \mathcal{U}$ be non UA, and let b be the number of its breakpoints in $]T, T + d]$. Then, if $p \in \mathbb{N}$, $p > 1$, is a divisor of $S^{\mathcal{P}}$, it is also a divisor of b.*

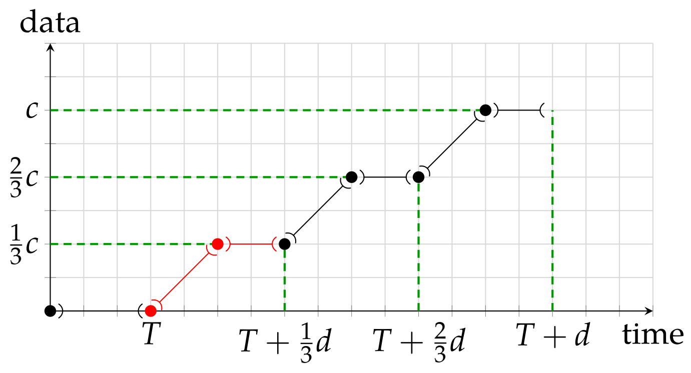
Proof. Let $\tilde{d}$ be the minimal period for f, and let $\tilde{b}$ be the number of breakpoints in $\left]T, T + \tilde{d}\right]$. Now, by definition, $d = p \cdot \tilde{d}$ for some $p \in \mathbb{N}$. By construction, then, if $p > 1$, $S^{\mathcal{P}}$ consists of p matching parts, hence p is a divisor of $S^{\mathcal{P}}$. By Equation (3.1), if $t_b \in \left]T, T + \tilde{d}\right]$ is a breakpoint, then $t_b + \tilde{d}$ is also a breakpoint. Then, it is $b = p \cdot \tilde{b}$, i.e., p is also a divisor of b. □

For instance, in Figure 12.3b, $p = 3$ is a divisor of $S^{\mathcal{P}}$, and there are in fact 6 breakpoints in $]T, T + d]$. Lemma 12.3 states that, in order to minimize the period, we can limit ourselves to testing if *the divisors of* b are also divisors of $S^{\mathcal{P}}$. This allows us to formulate an efficient algorithm that minimizes the period d of a representation R_f:

- Count b as the number of breakpoints of f in $]T, T + d]$;
- Find the prime factorization of b;
- Exhaustively test if the prime factors of b are also divisors of $S^{\mathcal{P}}$: if they are, update $S^{\mathcal{P}}$, d and c accordingly.



(a) f with breakpoints in $]T, T+d]$ highlighted as circles.

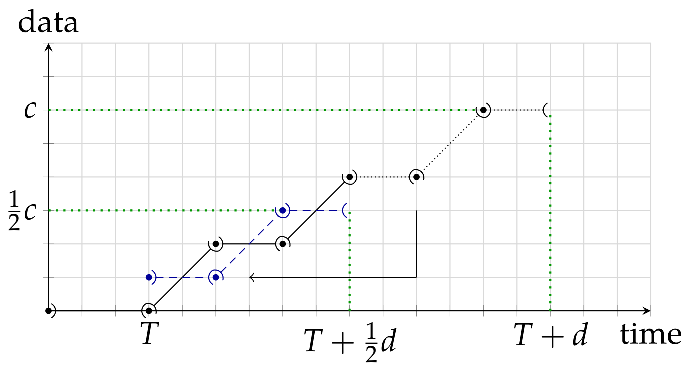


(b) Factorization of $S^{\mathcal{P}}$ with $p = 3$. We can replace d, c and $S^{\mathcal{P}}$, defined in $]T, T+d]$, with, respectively, $\frac{d}{3}$, $\frac{c}{3}$, and $S^{\mathcal{P}'}$, defined in $\left]T, T+\frac{d}{3}\right]$ and highlighted in red. The latter is an equivalent, but more compact, representation of f.

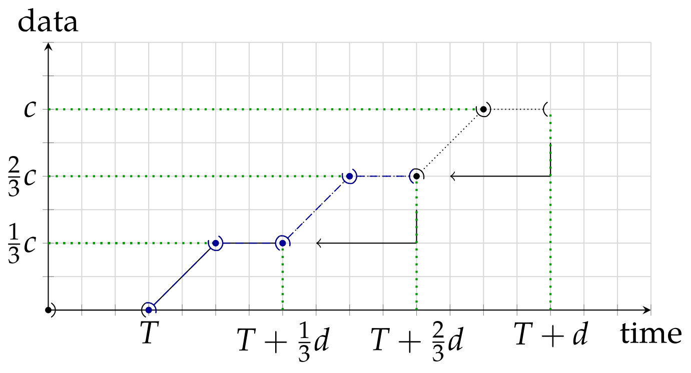
Figure 12.3: Example of factorization of the pseudo-periodic part.

We exemplify this algorithm on the function in Figure 12.3. From Figure 12.3a we observe that $b = 6$, whose prime factorization is $2 \cdot 3$. Therefore, we test these two primes as possible divisors of $S^{\mathcal{P}}$, as shown in Figure 12.4. Testing a factor p consists, in general, in dividing the sequence in p parts, defined in $\left]T + i \cdot \frac{d}{p}, T + (i+1) \cdot \frac{d}{p}\right]$ for $i = 0 \ldots p-1$, and checking whether, after shifting them down by $\frac{d}{p}$ and left by $\frac{c}{p}$, they all match. Figure 12.4a shows that the test with $p = 2$ fails, whereas Figure 12.4b shows that the test with $p = 3$ succeeds. After a division succeeds, it is convenient to immediately replace $S^{\mathcal{P}}$, d and c with their smaller equivalents $S^{\mathcal{P}'}$, $\frac{d}{p}$ and $\frac{c}{p}$, so that the upcoming tests with other factors of b are more efficient. In particular, $n\left(S^{\mathcal{P}'}\right) < n\left(S^{\mathcal{P}}\right)$. The test is run exhaustively for all prime factors of b. If a prime factor p has multiplicity $m > 1$, it is tested as a divisor of $S^{\mathcal{P}}$ up to m times.

Obtaining number b requires counting breakpoints of f in $]T, T+d]$, which is a simple $\mathcal{O}\left(n\left(S^{\mathcal{P}}\right)\right)$ check. To find the prime factorization of b, we will need the prime numbers in $2 \ldots \sqrt{b}$. Computing primes until a given x is something that can be done offline – we use an offline list of 1000 primes in our implementation, which



(a) Test of prime factor 2. The last half is translated (down by $\frac{1}{2}c$, left by $\frac{1}{2}d$) on top of the first half. Since they do not match, 2 is not a divisor of $S^{\mathcal{P}}$.



(b) Test of prime factor 3. The last third is translated (down by $\frac{1}{3}c$, left by $\frac{1}{3}d$) on top of the second one, and then again on top of the first one. As they all match, 3 is a divisor of $S^{\mathcal{P}}$.

Figure 12.4: Example of the factorization algorithm.

is enough for periods exceeding 62 millions. Lastly, testing if a prime factor p is a divisor of $S^{\mathcal{P}}$ entails a linear comparison between its parts. Let n_p be the number of prime divisors of b. In the worst-case, the algorithm will test, unsuccessfully, all n_p divisors, thus the complexity of this last step is $\mathcal{O}\left(n_p \cdot n(S^{\mathcal{P}})\right)$.

12.2 Minimization of the transient

In a non-minimal representation, the period start T can be overestimated, making the transient part longer than strictly necessary. This algorithm aims at removing this excess transient by bringing forward the period start. We exemplify this process starting from the representation in Figure 12.5.

As a first step, which we call *by-period*, we check if the rightmost end of the transient part contains sequences that match with the (already minimized) periodic part itself – and remove them, in case. In the example of Figure 12.5 we can see that the

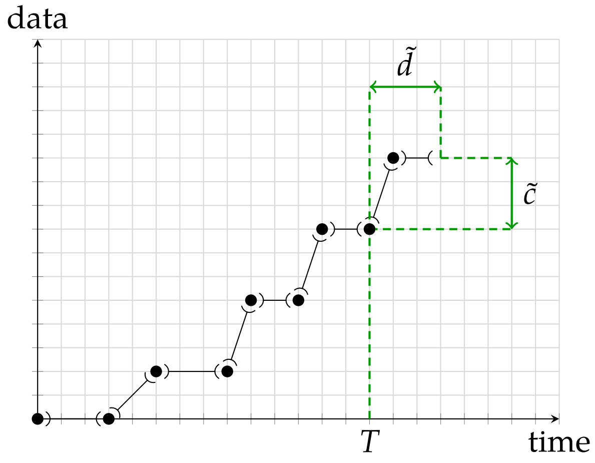


Figure 12.5: Example of a non-minimal representation.

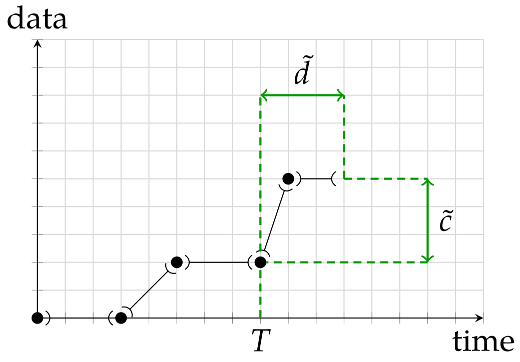


Figure 12.6: Representation reduced by a whole number of periods.

representation is equivalent to the one in Figure 12.6. We can obtain this result algorithmically by comparing the sequence in $]T, T+\tilde{d}]$ with the transient sequence immediately before, i.e., in $]T-\tilde{d}, T]$. If the two are matching, then the period start can be brought forward to $T' = T - \tilde{d}$, while the other parameters stay the same. This operation removes a period's worth of elements from $S^{\mathcal{T}}$. We repeat this process iteratively until the comparison fails. The end result is a reduction of the representation by a number of periods $k \in \mathbb{N}_0$, and an earlier period start $T' = T - k \cdot \tilde{d}$.

As a second step, which we call *by-segment*, we test if *parts* of a period, instead of whole periods, can be found at the right end of the transient part. In the example we can see that the representation of Figure 12.6 is equivalent to the one in Figure 12.7. We can algorithmically obtain this result by comparing the last pair (point, segment) of the periodic part, say in $]T+\tilde{d}-l, T+\tilde{d}]$, thus of length l, with the transient part of the same length immediately before the period start, thus in $]T-l, T]$. If the two are matching, then the period start is brought forward to $T' = T - l$, while the other parameters stay the same.

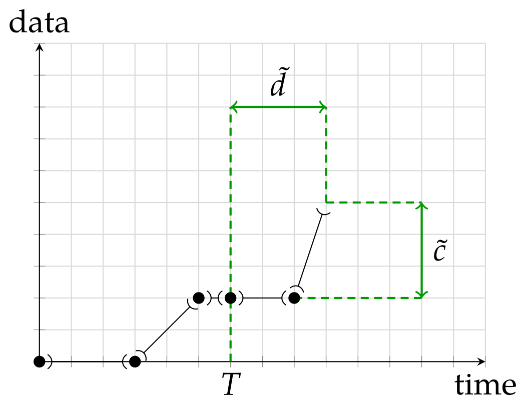


Figure 12.7: Representation reduced by a segment, altering the pseudo-period sequence.

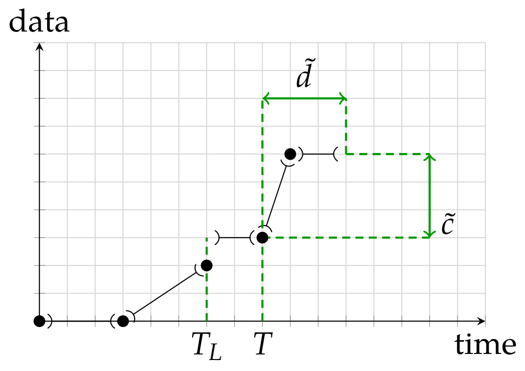


Figure 12.8: Example of function with a right-discontinuity before the period start: T_L is an infimum for T, but not its minimum.

While this appears close to the *by-period* step, an important difference is that $S^{\mathcal{P}}$ needs also be altered as a result (although $\tilde{d}$ will remain the same).

The above steps are repeated until no further changes can be made. Transient minimization can also be applied to UA functions. For these, one should just check if the tail of the transient is aligned with the period half-line.

Our transient minimization algorithm always achieves the minimum T, if one exists. In fact, some functions do not admit a *minimum* T. Figure 12.8 shows a function which is not right-continuous in T_L, and whose periodic part must start *after* T_L (recall that Equation (3.1) includes a *weak* inequality), hence admits no *minimum* T. In this particular case, our by-segment step yields the representation in Figure 12.8. We observe that using any T' in $]T_L, T]$, by removing a fragment of the rightmost segment in the periodic part, would yield an equivalent representation with the same $n(S)$ anyway.

Regarding the algorithmic complexity of this algorithm, note that the linear check involves, at most, the entire $S^{\mathcal{T}}$, then the complexity is $\mathcal{O}\left(n(S^{\mathcal{T}})\right)$. Thus the cost of the entire representation minimization algorithm is $\mathcal{O}\left(n(S^{\mathcal{T}}) + n_p \cdot n(S^{\mathcal{P}})\right)$.

12.3 Performance evaluation

In this section, we discuss the results obtained by implementing this algorithm within *Nancy*.

The algorithm is implemented via the `Curve.Optimize()` method, which runs the two algorithms discussed above, implemented internally as `Curve.PeriodFactorization()` and `Curve.TransientReduction()`. This method is called automatically by most methods before returning their result, if the `settings` object passed has `UseRepresentationMinimization` set to `true`. Note that this is the default setting, hence most users will experience the performance benefits shown below without further actions.

12.3.1 Impact on flow control use case

We first show the impact in the example we presented in Section 6.1. We repeated the same computations, this time adding representation minimization in between each operation (both minima and convolutions) – i.e., with `UseRepresentationMinimization` set to `true`.
The new results are in Table 12.1 and Table 12.2, which highlight both speedups and reductions in representation size up to three orders of magnitude.
An important aspect that links both is that a larger representation size translates directly to a higher memory occupancy during computations. As the occupied memory approaches the maximum allowed by the computer system, the performance is also affected. We do believe that the reason why some SACs do not terminate (within a reasonable time) with the exact method is that they end up occupying all the available memory, hence disk swaps start kicking in.

Table 12.1: Computational results, exact method.

	unoptimized results	with minimization
comp. time of $\overline{\beta_2 \otimes \beta_3^{eq} + W_3}$	6 h 24 m	6 s
$n\left(\beta_2 \otimes \beta_3^{eq} + W_3\right) \to n\left(\overline{\beta_2 \otimes \beta_3^{eq} + W_3}\right)$	$10 \to 10600$	$10 \to 10$
comp. time of $\overline{\beta_1 \otimes \beta_2^{eq} + W_2}$	> 24 h	13 s
$n\left(\beta_1 \otimes \beta_2^{eq} + W_2\right) \to n\left(\overline{\beta_1 \otimes \beta_2^{eq} + W_2}\right)$	*unknown*	$14 \to 6$

Table 12.2: Computational results, approximate method.

	unoptimized results	with minimization
comp. time of $\beta_{\{1,3\}} = \overline{\beta_1 \otimes \beta_2 + W_2} \otimes \overline{\beta_2 \otimes \beta_3 + W_3}$	0.14 s	0.11 s
$n(\overline{\beta_1 \otimes \beta_2 + W_2}), n(\overline{\beta_2 \otimes \beta_3 + W_3}) \to n\left(\beta_{\{1,3\}}\right)$	$6, 6 \to 270$	$6, 6 \to 42$
comp. time of $\beta_{\{1,4\}} = \beta_{\{1,3\}} \otimes \overline{\beta_3 \otimes \beta_4 + W_4}$	6 h 13 m	13.47 s
$n\left(\beta_{\{1,3\}}\right), n(\overline{\beta_3 \otimes \beta_4 + W_4}) \to n\left(\beta_{\{1,4\}}\right)$	$270, 6 \to 1456$	$42, 6 \to 6$

12.3.2 Extended study

In this study, we use curves of the form $\beta_{R,\theta,h} = \overline{\beta_{R,\theta} + h}$, where $\beta_{R,\theta}$ is a rate-latency curve, with latency θ and rate R, and h is the ordinate of a constant function. We consider, in each experiment, the convolution of three of such curves, $(\beta_a \otimes \beta_b) \otimes \beta_c$, whose parameters are randomly generated.

First, we computed the convolutions without any improvement. Then, we computed the same convolutions using representation minimization both on the results and in between any intermediate step: for instance, when we compute Equation (3.11), we minimize the result of each of the four partial convolutions. Note that the algebraic properties discussed in Chapter 13 were not used in this study.

We first report in Figure 12.9 the reduction in the number of elements of the result. Each experiment is reported as a point on a Cartesian plane (with logarithmic scales): its ordinate is the number of elements of the minimized result, and its abscissa is the number of elements of the unoptimized one. The dashed line is the bisector, below which the representation reduction factor is larger than 1. This makes it easier to visualize the order of magnitude of the reduction, which is in fact the horizontal (or vertical) distance between a point and the bisector. Figure 12.9 highlights representation reductions of one to three orders of magnitude. A box plot of the representation reduction is shown in Figure 12.10.
In the previous section, we have shown that in the flow control example the representation minimization also yielded a speedup of orders of magnitude. We therefore evaluate the cost of the above operations in Figure 12.11, which compares the optimized and unoptimized runtimes, and in Figure 12.12, which reports a box plot of the reduction in computation times.

Our results show that – while some gains are certainly there in most cases – a high reduction in the state occupancy does not always yield a similar reduction in computation times. The median time reduction is in the order of 10%. This can be explained by observing that the impact of *period minimization* on the *lcm* is variable. Consider for example two curves, f and g, and their respective representations with $d_f = 30$ and $d_g = 30$, such that by performing period minimization on these representations we obtain $\tilde{d}_f = 5, \tilde{d}_g = 6$. While the reduction in size is noticeable, the

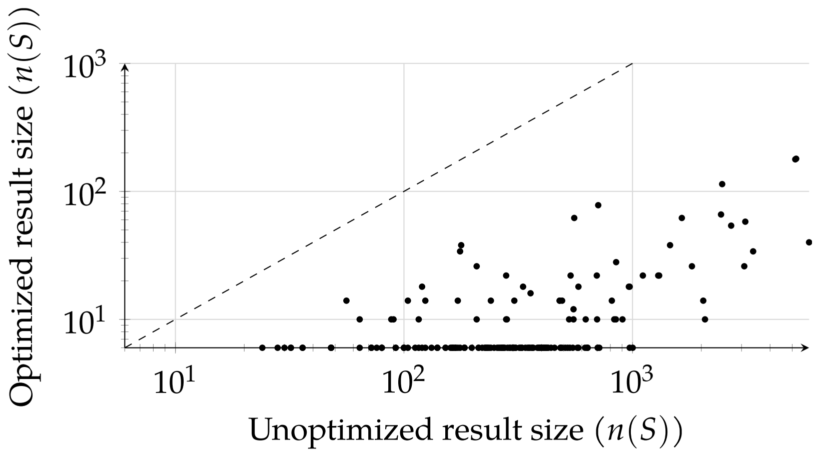


Figure 12.9: Cardinality of the results of the convolution of three subadditive functions, with and without minimization.

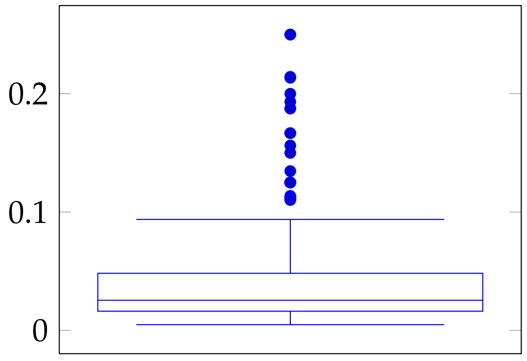


Figure 12.10: Convolution of three subadditive functions: reduction of the cardinality of the result due to minimization.

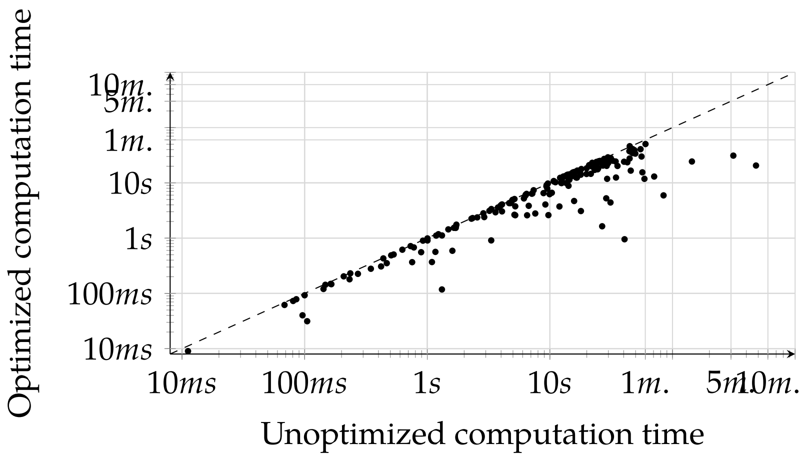


Figure 12.11: Computation times of the convolution of three subadditive functions, with and without minimization.

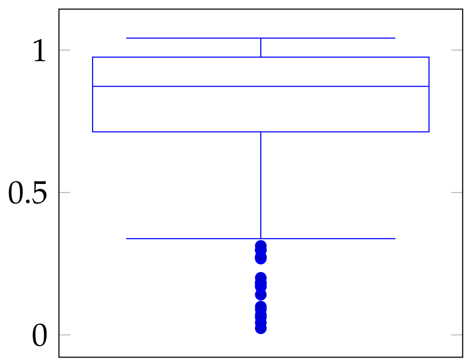


Figure 12.12: Convolution of three subadditive functions: reduction of computation times due to minimization.

same cannot be said about computing $f \otimes g$, since $\text{lcm}(30, 30) = \text{lcm}(5, 6)$. Thus, the more substantial speedups are obtained when minimization succeeds in removing a common factor from both operands (e.g., factor 5 from both f and g). Note that these cases (to which the example of Section 12.3.1 belongs) depend on numerical properties of the operands, hence are hard to obtain using random generation of the input (but not impossible – check the outliers at the bottom of Figure 12.12).

We stress that the most remarkable benefits of representation minimization lie in enabling the computation of SACs, hence the analysis of flow control networks via the exact method (Equation (2.8)). In this case, representation minimization is indispensable, since the complexity of the SAC algorithm is exponential with the number of elements. Unfortunately, we are not able to produce a speedup figure for the SACs, since unoptimized SACs with random parameters hardly ever terminate at all.

On the other hand, the above experiments highlight that applying minimization always yields a significant size reduction, which helps with memory management; it yields at least a moderate time reduction, most of the time, and – even in the rare cases when it fails to provide a speedup – the time spent on applying it is negligible.

Chapter 13

Improving (min,+) Convolution of Subadditive Curves

The work described in this chapter is part of the contributions in [ZS23]*, coauthored with Giovanni Stea. We wish to thank Paul Nikolaus for the help on improving the formalization of this work.*

As we presented in Section 3.4, in a (min,+) convolution the resulting period grows like the least common multiple (lcm) of the period of the operands, and the complexity of the by-sequence algorithm is superquadratic with the length of the sequences. Thus, it is possible to find instances where a *single* (min,+) convolution may take very long.
It is however possible to leverage algebraic properties to reduce the complexity of this operation. It is already well known, for example, that the (min,+) convolution of concave curves is equal to their minimum. In this chapter, we discuss similar properties for subadditive functions, that have a large on the complexity of their (min,+) convolution. To the best of our knowledge, this has never been observed before.

Before continuing, we add a note on the symmetrical property to subadditivity, i.e., superadditivity. In principle, we believe that similar properties as the ones discussed in this chapter may be obtained for (max,+) convolution of superadditive functions and, therefore, for the computation of superadditive closure. The only reason we did not include such result in this thesis is lack of time to properly go over the related formalization and practical experiments – this is shared to the many other paths of future research discussed in Chapter 16.

13.1 Improving convolution of subadditive functions with dominance

We first observe that *dominance* can be leveraged to abate the complexity of convolutions. We say that g *dominates* f if $g(t) \geq f(t)\ \forall t \geq 0$. In this case, the following property applies.

Theorem 13.1 (Convolution of subadditive functions with dominance)**.** *Let f and g be functions $\in \mathcal{U}$ such that $g(0) = 0$, $g(t) \geq f(t)\ \forall t$, f is subadditive, and $f \otimes g$ is well-defined.*[1] *Then,*

$$f \otimes g = f. \tag{13.1}$$

Proof. Since f is subadditive, $f(u) + f(s) \geq f(u+s)\ \forall u, s$. Then, for any $t = u + s$, $f(u) + g(s) \geq f(u) + f(s) \geq f(t)$. Thus, $(f \otimes g)(t) = \inf_{u+s=t}\{f(u) + g(s)\} = f(t) + g(0) = f(t)$. □

In order to apply this theorem algorithmically, we first need to compare f and g. Dominance can be verified by checking statements $f = f \wedge g$, $g = f \wedge g$. Both the minimum and the equivalence check have linear costs as discussed in Section 9.2. When either is true, Theorem 13.1 allows us to bypass the convolution altogether, which is instead superquadratic. Note that this theorem (as well as the following one) requires only the *dominated* function f to be subadditive, whereas the dominant function g can have any shape, as long as $g(0) = 0$.

When dominance does not hold, we can test a weaker property, *asymptotic dominance*. We say that g *dominates f asymptotically* if it exists $t^* > 0$ such that $g(t) \geq f(t)\ \forall t \geq t^*$. Note that $\rho_g > \rho_f$ is a sufficient condition for this to occur, but not a necessary one. In this case, we can resort to a "simpler" convolution as follows.

Theorem 13.2 (Convolution of subadditive functions with asymptotic dominance)**.** *Let f and g be functions $\in \mathcal{U}$ such that $f(0) = g(0) = 0$, $g(t) \geq f(t)\ \forall t \geq t^*$, f is subadditive and $f(t) > -\infty\ \forall t \geq 0$.*

Let $g = g_a \wedge g_b$ be a decomposition of g where

$$g_a(t) = \begin{cases} g(t) & \text{if } t \in [0, t^*[, \\ +\infty & \text{if } t \geq t^*, \end{cases} \qquad g_b(t) = \begin{cases} 0 & \text{if } t = 0, \\ +\infty & \text{if } t \in]0, t^*[, \\ g(t) & \text{if } t \geq t^*. \end{cases}$$

Then

$$f \otimes g = f \otimes g_a \wedge f.$$

[1] As mentioned in [BT08, p. 7] and recalled in Chapter 3, convolution $f \otimes g$, with $f, g \in \mathcal{U}$, is not defined if there exist t_1, t_2 such that $f(t_1) = +\infty$ and $g(t_2) = -\infty$, or vice versa (i.e., both are infinite, with opposite signs).

Proof. Decompose g as per the hypothesis. Then

$$\begin{aligned} f \otimes g &= f \otimes (g_a \wedge g_b) \\ &= f \otimes g_a \wedge f \otimes g_b. \end{aligned}$$

For the latter part, we observe that $g_b \geq f \; \forall t$, and that $g_b(0) = 0$. We can therefore apply Theorem 13.1, for which $f \otimes g_b = f$. Thus

$$\begin{aligned} f \otimes g &= f \otimes g_a \wedge f \otimes g_b \\ &\overset{(13.1)}{=} f \otimes g_a \wedge f. \end{aligned}$$

□

If g dominates f only asymptotically, then f is above g at some point, but will eventually fall below it. Accordingly, there exists t^* such that $f(t) \leq g(t) \; \forall t \geq t^*$, and by algorithmic construction of $f \wedge g$ we can say that $T_{f \wedge g}$ is in fact such t^*.[2] Therefore, we can apply Theorem 13.2, and compute $f \otimes g$ by:

- Computing $h = f \otimes g_a$. Since $g_a(t) = +\infty \; \forall t \geq t^*$, computing this convolution will involve $d = d_f$ rather than $d = \mathrm{lcm}(d_f, d_g)$, thus smaller D and S_f^I, reducing the cost of computation.

- Computing $f \otimes g = h \wedge f$. Being a minimum, it has a linear cost, but again $d = \mathrm{lcm}(d_h, d_f) = \mathrm{lcm}(d_f, d_f) = d_f$, hence the number of operations is greatly reduced.

The main benefit of applying Theorem 13.2 lies in dispensing with computing the representation of f and g over a possibly very long period $d = \mathrm{lcm}(d_f, d_g)$.

If neither of the above theorems can be applied, we can resort to the following result.

Theorem 13.3 (Convolution of subadditive functions as self-convolution of the minimum)**.** *Let f and g be subadditive functions $\in \mathcal{U}$ such that $f(0) = g(0) = 0$, and $f \otimes g$ is well-defined. Then,*

$$f \otimes g = (f \wedge g) \otimes (f \wedge g).$$

Proof. We recall that if f is subadditive with $f(0) = 0$, then $f \otimes f = f$ (Equation (4.1) in Corollary 4.7); and if $f(0) = g(0) = 0$, then $f \wedge g \geq f \otimes g$ (Equa-

[2] The alert reader may note that this is not true if *minimization of the transient* is applied to $f \wedge g$ – we indeed back up $T_{f \wedge g}$ beforehand.

tion (4.3) in Lemma 4.8). Then

$$\begin{aligned}(f \wedge g) \otimes (f \wedge g) &= (f \otimes f) \wedge (f \otimes g) \wedge (g \otimes f) \wedge (g \otimes g) \\ &\overset{(4.1)}{=} f \wedge (f \otimes g) \wedge (f \otimes g) \wedge g \\ &= f \wedge g \wedge (f \otimes g) \\ &= (f \wedge g) \wedge (f \otimes g) \\ &\overset{(4.3)}{=} f \otimes g.\end{aligned}$$

□

To exploit this theorem, we would first need to compute $f \wedge g$. However, this computation is also a prerequisite for testing Theorem 13.1, which one would try first anyway. Theorem 13.3 transforms a convolution into a *self-convolution*. Self-convolutions can be computed more efficiently than standard convolutions. In fact, we can bypass more than half of the elementary convolutions within the by-sequence algorithm, as per the following properties:

Proposition 13.4 (Avoiding duplicates in self-convolutions)**.** *A self-convolution $h \otimes h$, $h \in \mathcal{U}$, can be computed through a single by-sequence convolution with $S_h^I \otimes S_h^I$, with $D = [0, 2 \cdot T_h + 2 \cdot d_h[$.*
Since this by-sequence convolution is symmetric, we can reduce the number of its elementary convolutions to

$$\frac{n^2 - n}{2} < n^2,$$

where $n = n(S_h^I)$.

Proof. Since the two operands of the convolution have the same ρ_h, from [BT08] we know that

- $T = T_h + T_h + d = 2 \cdot T_h + d^3$;
- $d = \operatorname{lcm}(d_h, d_h) = d_h$;
- $c = \rho \cdot d = c_h$.

Consider then $S_h^I, D = [0, T + d[= [0, 2 \cdot T_h + 2 \cdot d_h[$, and its composing elements e_i, $1 \leq i \leq n$. It is:

$$\begin{aligned}S_h^I &= e_0 \wedge e_1 \wedge \cdots \wedge e_n; \\ S_h^I \otimes S_h^I &= \bigwedge_{e_i, e_j} e_i \otimes e_j.\end{aligned}$$

[3]When combined with Theorem 13.3, we can use $T = \min(2 \cdot T_{f \wedge g}, T_f + T_g)$.

The by-sequence convolution entails n^2 elementary convolutions. However, convolution being commutative, many of these are computed twice, e.g., $e_1 \otimes e_2$ and $e_2 \otimes e_1$. This can be avoided by computing instead

$$\begin{aligned} S_h^I \otimes S_h^I &= \bigwedge_{e_i,e_j:i,j\in[0..n-1]} e_i \otimes e_j \\ &= \left(\bigwedge_{e_i,e_j:i,j\in[0..n-1],i<j} e_i \otimes e_j \right) \wedge \left(\bigwedge_{e_i:i\in[0..n-1]} e_i \otimes e_i \right), \end{aligned}$$

which results in $n + (n-1) + \cdots + 1 = \frac{n^2-n}{2}$ elementary convolutions. □

Therefore, in a self-convolution (such as the one of Theorem 13.3) one can halve the number of elementary convolutions. On top of that, a further improvement is warranted by the following proposition:

Proposition 13.5 (Reducing the number of convolutions by element coloring)**.** *Let f and g be subadditive functions $\in \mathcal{U}$ such that $f(0) = g(0) = 0$, and $h = f \wedge g$ (thus, $h(0) = 0$). Let S_h^I be the sequence necessary to compute $h \otimes h$. Given* colors $\{f, g\}$, *for an element $e_k \in S_h^I$ defined in interval I_k, we define its color as*

$$color(e_k) = \begin{cases} f & \text{if } e_k(t) = f(t)\ \forall t \in I_k, \\ g & \text{otherwise.} \end{cases}$$

An element's color is thus the function (f or g) it belongs to. Then, we can compute $S_h^I \otimes S_h^I$ as:

$$S_h^I \otimes S_h^I = \left(\bigwedge_{\substack{e_i,e_j\in S_h^I \\ color(e_i)\neq color(e_j)}} e_i \otimes e_j \right) \wedge S_h^I, \tag{13.2}$$

i.e., we can omit computing elementary convolutions of elements of the same color.

Proof. Since $h(0) + h(t) = h(t)$, we can write the convolution as

$$\begin{aligned} (h \otimes h)(t) &= \inf_{0\le s\le t} \{h(s) + h(t-s)\} \\ &= \inf_{0<s<t} \{h(s) + h(t-s)\} \wedge h(t). \end{aligned}$$

Thus, we can ignore in the computation any pair (t_i, t_j), such that $t_i + t_j = t$, for which $h(t_i) + h(t_j) \ge h(t)$. We show that elements of the same color fall in such category.

Let e_i, e_j be elements of S_h^I defined, respectively, on intervals I_{e_i}, I_{e_j} and such that $color(e_i) = color(e_j) = f$. Let $I_{e_i\otimes e_j} = \left\{ t = t_i + t_j \mid t_i \in I_{e_i}, t_j \in I_{e_j} \right\}$.

Then, for any $t \in I_{e_i \otimes e_j}$ and $t_i \in I_{e_i}, t_j \in I_{e_j}$ such that $t = t_i + t_j$, we have that

$$\begin{aligned}(h \otimes h)(t) &\leq h(t_i) + h(t_j)\\ &= f(t_i) + f(t_j),\end{aligned}$$

since $color(e_i) = color(e_j) = f$. On the other hand, due to subadditivity of f,

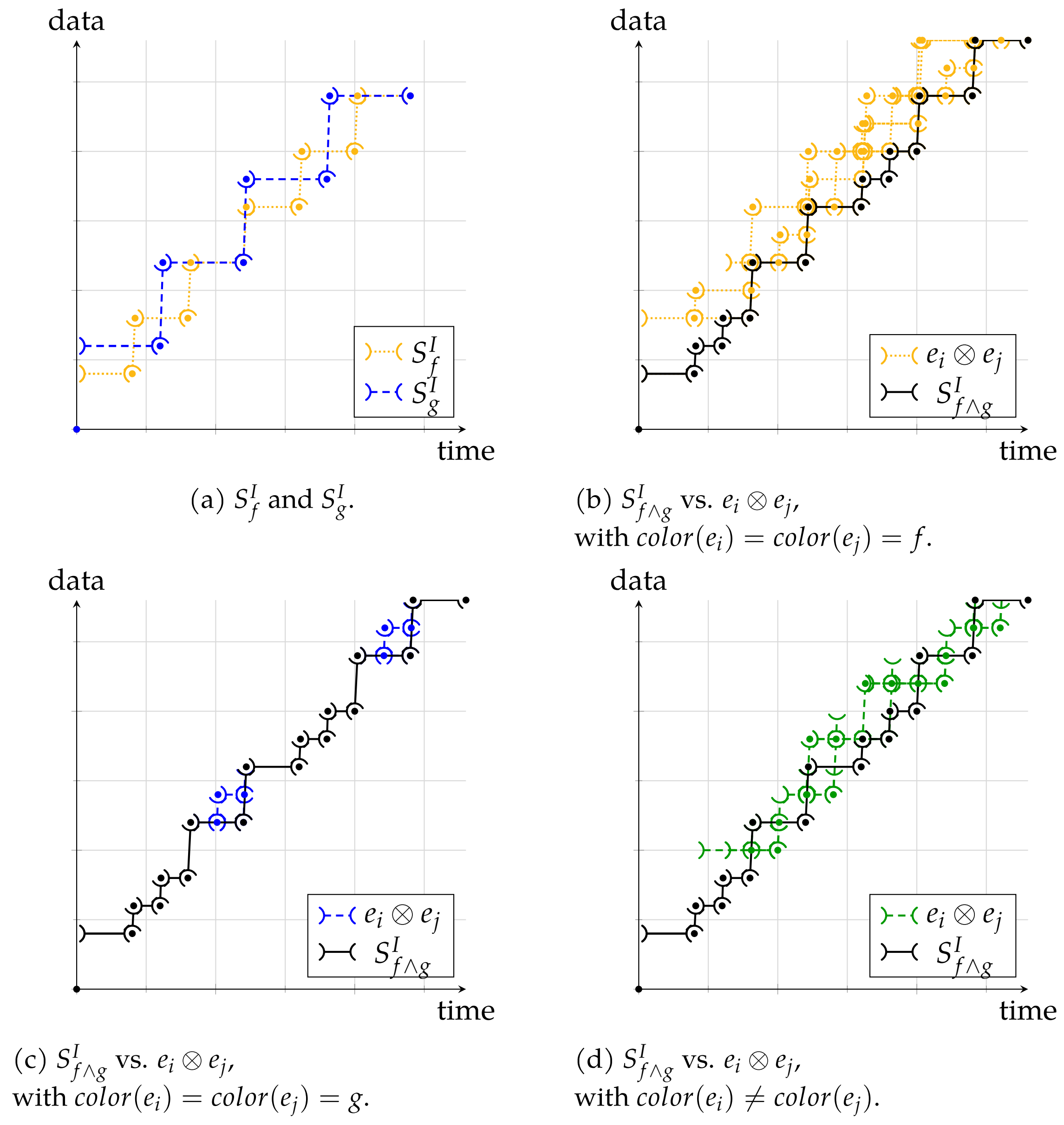
$$f(t_i) + f(t_j) \geq f(t) \geq h(t).$$

Thus, $(e_i \otimes e_j)(t) \geq h(t)$. Obviously, the same holds if $color(e_i) = color(e_j) = g$.

Therefore, in order to compute $S^I_h \otimes S^I_h$, it is sufficient to include in the computation of the lower envelope the sequence S^I_h and the convolutions of elements with *different* colors, hence Equation (13.2). □

The idea behind Proposition 13.5 can be visualized through the example in Figure 13.1. Take f and g (Figure 13.1a), which intersect infinitely many times – hence their convolution cannot be simplified leveraging dominance. Figure 13.1b and Figure 13.1c report $S^I_{f \wedge g}$ against elementary convolutions $e_i \otimes e_j$, where e_i and e_j have the same color (f and g, respectively). These figures show that the results of these elementary convolutions are always above $f \wedge g$. Instead, in Figure 13.1d we see how convolutions of elements of *different* colors may yield elements below $f \wedge g$.

The above two properties allow one to make the computation of $(f \wedge g) \otimes (f \wedge g)$ as efficient as possible, skipping many elementary convolutions. However, it remains to be seen whether computing the above is faster than computing $f \otimes g$ directly. Our results, reported in Section 13.2.2, show that this is indeed the case in the vast majority of cases: the ensuing time reduction ranges from sizeable percentages to 10 times. Counterintuitively, this is not due to a reduction in the number of elementary convolutions (which is instead of the same order of magnitude in the two cases, despite the optimizations of Proposition 13.4 and Proposition 13.5). Rather, it is due to the different topological properties of the ensuing elements. A thorough discussion of this phenomenon is reported in Section 13.2.2.



(a) S^I_f and S^I_g.

(b) $S^I_{f \wedge g}$ vs. $e_i \otimes e_j$, with $color(e_i) = color(e_j) = f$.

(c) $S^I_{f \wedge g}$ vs. $e_i \otimes e_j$, with $color(e_i) = color(e_j) = g$.

(d) $S^I_{f \wedge g}$ vs. $e_i \otimes e_j$, with $color(e_i) \neq color(e_j)$.

Figure 13.1: Coloring example.

13.2 Performance evaluation

In this section, we show the impact of these optimizations, which were implemented in *Nancy*.

The improved algorithm is implemented through the `SubadditiveCurve` class, which inherits from `Curve` and overrides some of its methods. Namely, the `SubadditiveCurve` ⌋ `.Convolution()` method attempts to apply the optimizations of Theorems 13.1 to 13.3, checking if the operands are subadditive and if the assumptions of these theorems are verified. This behavior can be controlled via the `settings` argument: the optimizations are applied only if `UseSubadditiveConvolutionOptimizations` is set to `true` (it is, by default).

Note that – unless obtained via methods known to return a subadditive curve, such as `Curve.SubAdditiveClosure()`, a user has to actively indicate that a given curve is subadditive by using the `SubadditiveCurve` class instead of `Curve`. This is due to the fact that the test for subadditivity of a generic curve f, as discussed in Section 9.3, is implemented as the self convolution $f^\circ \otimes f^\circ$ followed by the test that this result is equal to f° again (see Lemma 9.3). Since this operation is not cheap in general, it would be detrimental to perform it without user interaction.

On the other hand, if the use case does involve subadditive curves marked as such with the `SubadditiveCurve` class, the user is not required to do anything more in order to benefit from the performance improvements shown here.

13.2.1 Impact on flow control use case

We first show the impact in the example we presented in Section 6.1. We repeated the same computations, this time exploiting also the theorems proved in this section. The new results are in Table 13.1 and Table 13.2, which highlight further reductions in computation time.

Table 13.1: Computational results, exact method.

	w/o optimizations	minimization	minimiz. + Th. 1,2,3
comp. time of $\overline{\beta_2 \otimes \beta_3^{eq} + W_3}$	6 h 24 m	6 s	0.47 s
$n\left(\beta_2 \otimes \beta_3^{eq} + W_3\right) \rightarrow n\left(\overline{\beta_2 \otimes \beta_3^{eq} + W_3}\right)$	$10 \rightarrow 10600$	$10 \rightarrow 10$	$10 \rightarrow 10$
comp. time of $\overline{\beta_1 \otimes \beta_2^{eq} + W_2}$	> 24 h	13 s	0.18 s
$n\left(\beta_1 \otimes \beta_2^{eq} + W_2\right) \rightarrow n\left(\overline{\beta_1 \otimes \beta_2^{eq} + W_2}\right)$	*unknown*	$14 \rightarrow 6$	$14 \rightarrow 6$

Table 13.2: Computational results, approximate method.

	w/o optimizations	minimization	minimiz. + Th. 1,2,3
comp. time of $\beta_{\{1,3\}} = \overline{\beta_1 \otimes \beta_2 + W_2} \otimes \overline{\beta_2 \otimes \beta_3 + W_3}$	0.14 s	0.11 s	0.09 s
$n(\overline{\beta_1 \otimes \beta_2 + W_2}), n(\overline{\beta_2 \otimes \beta_3 + W_3}) \rightarrow n\left(\beta_{\{1,3\}}\right)$	$6, 6 \rightarrow 270$	$6, 6 \rightarrow 42$	$6, 6 \rightarrow 42$
comp. time of $\beta_{\{1,4\}} = \beta_{\{1,3\}} \otimes \overline{\beta_3 \otimes \beta_4 + W_4}$	6 h 13 m	13.47 s	0.003 s
$n\left(\beta_{\{1,3\}}\right), n(\overline{\beta_3 \otimes \beta_4 + W_4}) \rightarrow n\left(\beta_{\{1,4\}}\right)$	$270, 6 \rightarrow 1456$	$42, 6 \rightarrow 6$	$42, 6 \rightarrow 6$

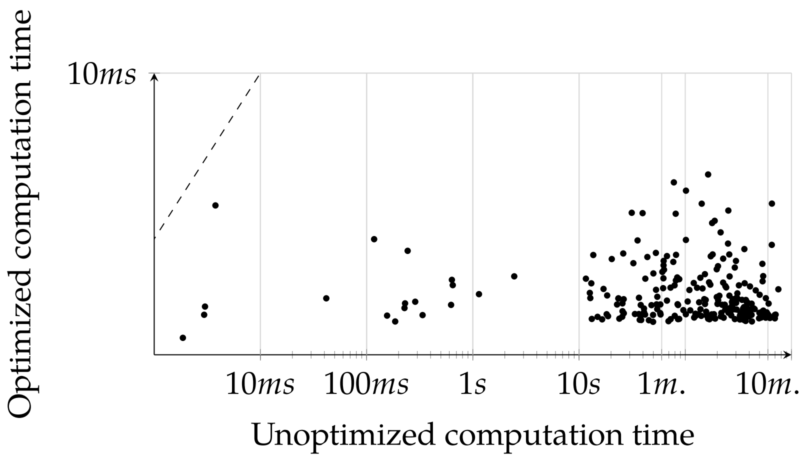


Figure 13.2: Results of the convolution of subadditive functions with dominance.

13.2.2 Extended study

Convolution of subadditive functions with dominance

We now test the impact of Theorem 13.1. We compute $\beta_{R_1,\theta_1,h_1} \otimes \beta_{R_2,\theta_2,h_2}$, where the operands are randomly generated and matching the hypotheses of Theorem 13.1. To make the comparison more insightful, in these and the following experiments we apply representation minimization to all intermediate computations in the baseline unoptimized algorithm.

The benefits of using Theorem 13.1 can be seen in Figure 13.2, which clearly shows that most speedups are in the region of 10^5 times. In many cases the unoptimized convolution lasted more than 10 minutes, while the optimized version seldom lasted more than 1 ms. This means that dominance is a property worth checking.

Convolution of subadditive functions with asymptotic dominance

We compute $\beta_{R_1,\theta_1,h_1} \otimes \beta_{R_2,\theta_2,h_2}$, where the operands are randomly generated and matching the hypotheses of Theorem 13.2. The impact of Theorem 13.2 is shown in Figure 13.3, which still highlights speedups in the order of 10^5 times: unoptimized convolutions taking several minutes are often reduced to fractions of a second. However, some lengthy computations still take a sizable time even after the optimization. This is because the effect of Theorem 13.2 is to use the time of last intersection, rather than $\operatorname{lcm}(d_f, d_g)$, to determine the sequences to be convolved. In few cases, the former may exceed the latter, hence Theorem 13.2 may instead increase the cost (see the point above the bisector in the bottom-left corner of Figure 13.3). However, such cases are rare – and easy to avoid. In fact, we can compare the extremes of the cut intervals of the operands, computed with the standard algorithm and Theorem 13.2,

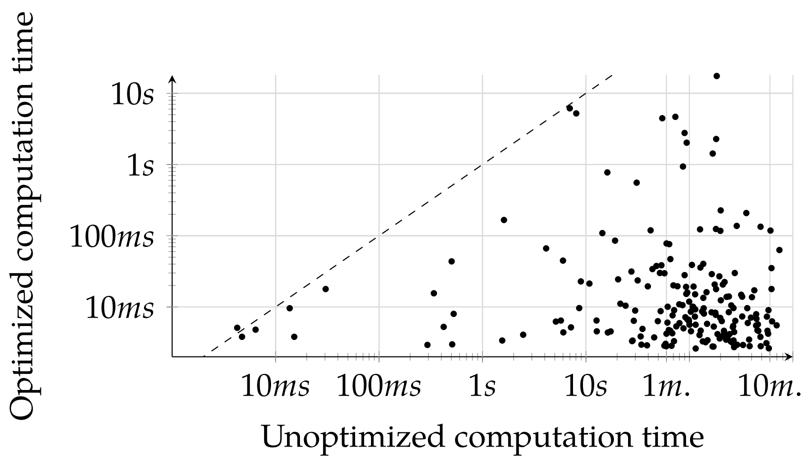


Figure 13.3: Results of the convolution between subadditive functions with asymptotic dominance.

and then run the algorithm that will involve fewer elementary convolutions.

Convolution of subadditive functions as self-convolution of the minimum

A first assessment of the impact of Theorem 13.3 (coupled with Propositions 13.4 and 13.5) is reported in Figure 13.4. It is evident from the figure that the speedup is less prominent in this case – the maximum that we get is 30 times. Note that the higher speedups are obtained when the unoptimized computations take more time (see the top-right cluster of points). However, there is a speedup in almost all cases – we only found one outlier at 0.99 times, meaning that using our theorem takes a little more time than using the basic convolution algorithm. The obtained speedup is mostly within one order of magnitude. For this reason, we report in Figure 13.5 a box plot of the *reduction of computation times* (which is the inverse of the speedup). With our method, computation times can be expected to be 30% to 80% of the unoptimized times.

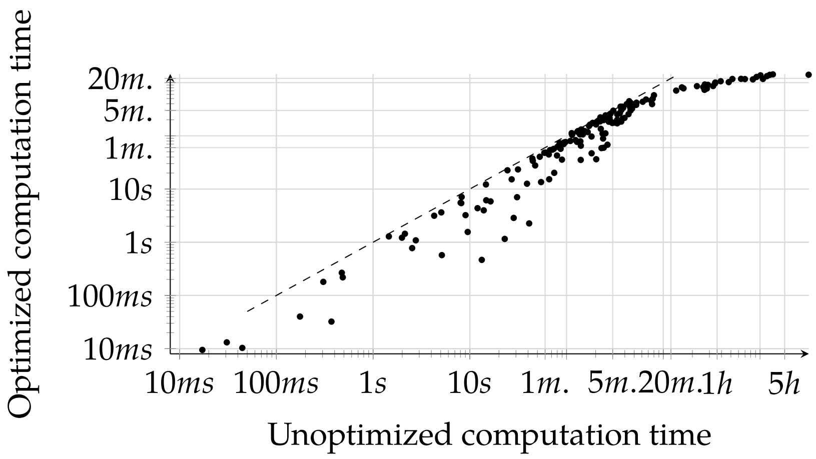


Figure 13.4: Results of the convolution between subadditive functions without asymptotic dominance.

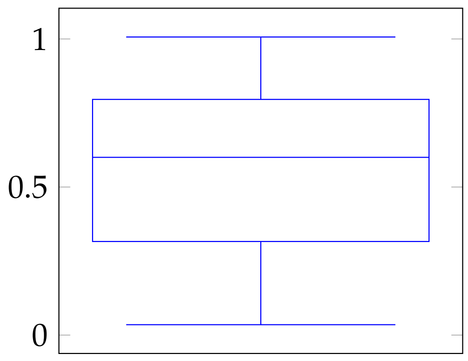


Figure 13.5: Reduction of computation times of the convolution between subadditive functions without asymptotic dominance.

Intuitively, one might expect the above speedup to be related to the number of elementary convolutions. However, Figure 13.6 shows that this is not the case: the number of elementary convolutions is roughly the same, regardless of the achieved speedup. In more than a few cases, applying Theorem 13.3 entails computing *more* elementary convolutions (i.e., all the points having abscissa smaller than 1), yet the optimized version yields a non-negligible speedup nonetheless.

The root cause of the speedups lies elsewhere. Recalling the lower envelope algorithm described in Section 8.8, its computation cost depends not only on the number of elements, but also depend on how much overlap there is between the elements. In fact, the more overlap there is between the elements of E, the higher the cost of this algorithm is. Some of the overlaps – actually, most – will not yield segments that end up being part of the lower envelope.

We show through a relevant example that computing $(f \wedge g) \otimes (f \wedge g)$ yields considerably less populated intervals than computing $f \otimes g$. The parameters are as

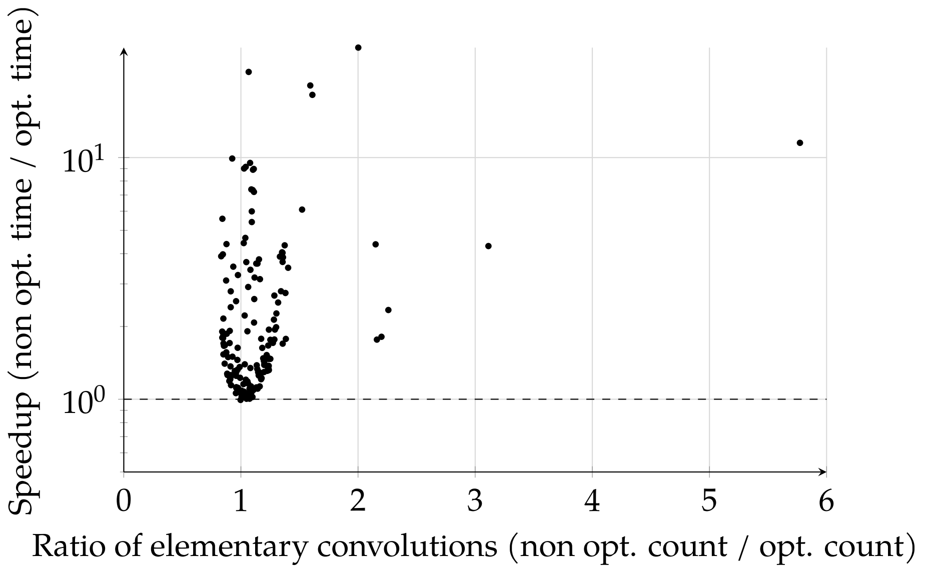


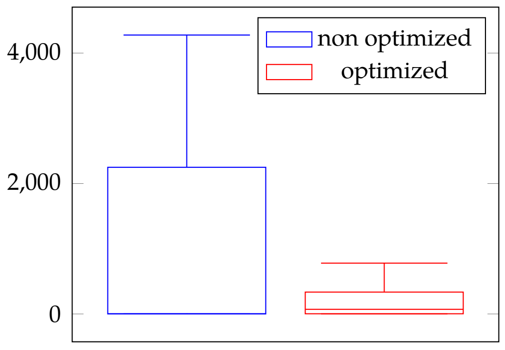
Figure 13.6: Ratio of elementary convolutions vs. speedup.

in Table 13.3.

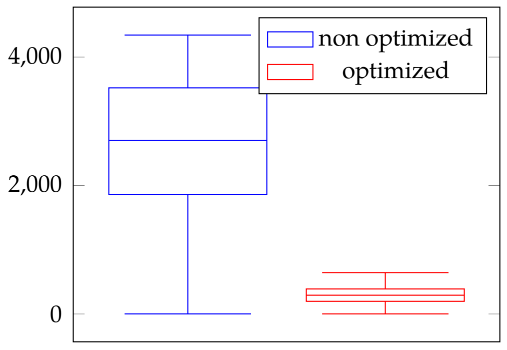
Table 13.3: Parameters of the example convolution

	R	θ	h
f	901	499	192
g	806	36	$\frac{6912}{499}$

In the non-optimized convolution algorithm, we need to compute the lower envelope of 810k elements, for which 220k intervals are used. In the optimized algorithm, we find instead 910k elements and 320k intervals. However, as Figure 13.7a highlights, there is a significant difference in how many intervals each element spans. This affects the cost of step 2, which takes 180s in the non-optimized algorithm vs. 42s in the optimized one. Moreover, as Figure 13.7b highlights, there is also a significant difference in how many elements a given interval list includes, which affects the cost of computing the per-interval lower-envelope. In fact, step 3 takes 370s in the non-optimized algorithm, against 70s in the optimized one.
Overall, applying the optimizations discussed produces, in this example, a fivefold speedup – which is counter-intuitive if one considers only the number of convolutions.



(a) Number of intervals per element.



(b) Number of elements per interval.

Figure 13.7: Relationship between elements and intervals, in a relevant example.

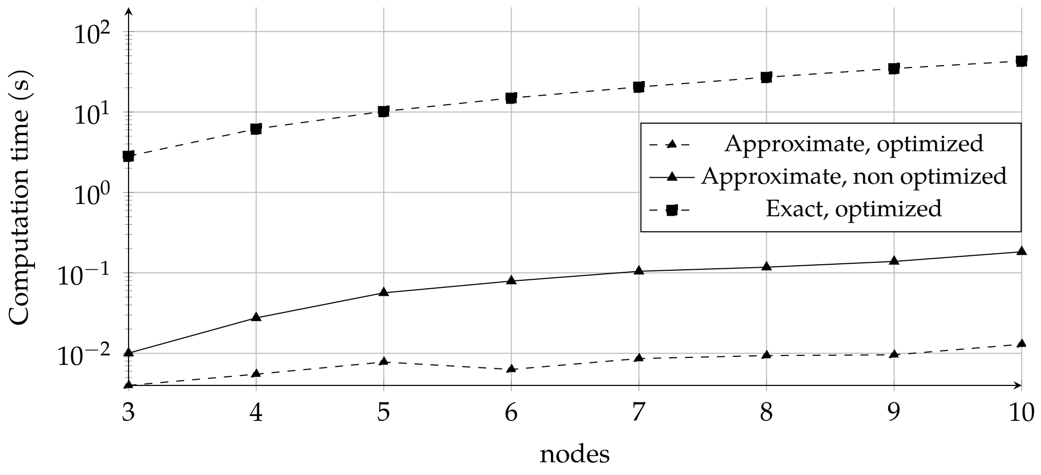


Figure 13.8: Performance comparison of the exact and approximate methods.

13.2.3 A case study

We now show how our method allows one to analyze flow-controlled networks. We consider a tandem of n nodes, $n = 2 \ldots 10$, where all nodes are described by the same rate-latency service curve $\beta_{16,2}$, and with input buffers of increasing size $W = 13, 15, \ldots, 29$. In Figure 13.8 we compare the runtimes of the exact method (2.8) and the approximate method (2.10), with and without the optimizations described in this thesis. We observe that the exact method can only be run *with* our optimizations: without them, the computations for a three-node tandem had not completed after 24 hours. The graph clearly shows that the approximate method is orders of magnitude faster than even the optimized exact one. However, our optimizations still take away one order of magnitude of computations in that as well. The experiments were run five times in independent conditions, and 95% confidence intervals were always within 1% of the average. For that reason, they are not reported in the graph.

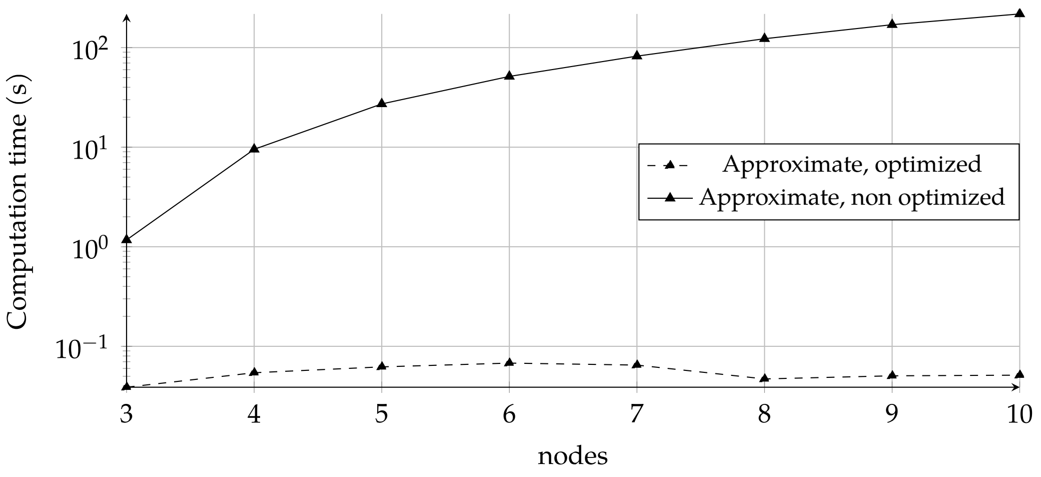


Figure 13.9: Performance comparison of the optimized/unoptimized approximate method.

We found that the computation times (whichever the method) are very sensitive to the actual parameters of the network: changing the numbers in the above example is likely to change the vertical scale of the above graph considerably. However, the same pattern still emerges: the unoptimized exact method is just unfeasible most of the times; the optimized exact method comes second; the approximate method is considerably faster, and even faster with our optimizations.
To support the above claim, we present another scenario in Figure 13.9, where the computation times for the approximate method are sensibly higher. To obtain such a difference, all it took was to modify rates to $R = 1600$, latencies to $\theta = 200$, and buffer sizes to $W = 1300, 1305, \dots, 1340$. In this case, the approximate method takes up to hundreds of seconds, whereas our optimizations curb the computations at fractions of a second. It is interesting to observe that our optimization yield times that are non monotonic with the tandem length (see, e.g., around $n = 8$). This is because a more favorable optimization kicks in at $n = 8$ and further abates computation times.

What our optimizations allow – for the first time, to the best of our knowledge – is an assessment of the *accuracy* of the approximate method. In fact, this requires being able to complete exact computations, which just cannot be done without these very optimizations (unless one handpicks very trivial scenarios and parameter values, with the obvious risk of undermining generality). Our results here are quite surprising. They show that the end-to-end service curves obtained via the approximate method are *always equal* to the exact ones. This occurs not only in the tandems described in this thesis, but in all the cases we analyzed, including many (several tens) with randomized configurations.

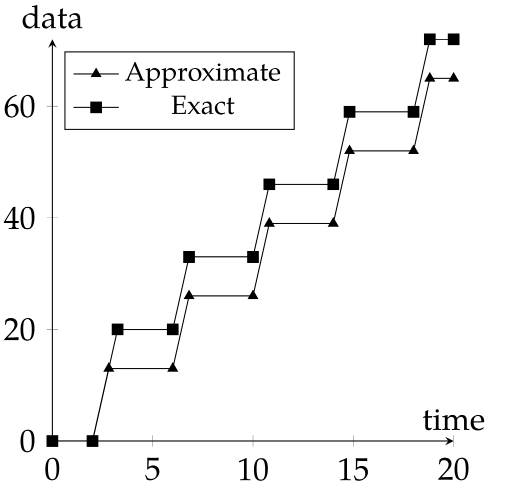
One may legitimately wonder if this is due to the fact that equality should hold in Equation (2.11), but so far no one was able to prove it. We show that this is not the case, i.e., there are cases when $\beta_i^{eq} > \beta_i^{eq'}$. Consider the three-node tandem in Section 2.3, and assume that nodes have the same rate-latency service curve $\beta_{16,2}$, and with input buffers $W_2 = 20$ and $W_3 = 13$.

When computing the equivalent service curve at the *first* node, i.e., β_1^{eq}, $\beta_1^{eq'}$, we obtain different results using the exact and approximate method, as shown in Figure 13.10a. It is $\beta_1^{eq} > \beta_1^{eq'}$. The difference can be explained by observing that, since $W_2 > W_3$, it is expected that the worst-case performance will be initially constrained by the larger buffer W_2 (see the first step of the exact β_1^{eq} in Figure 13.10a), then by the smaller buffer downstream (see the second step onwards of β_1^{eq}, in the same figure). The exact computation reflects this phenomenon, while the approximate method does not (see the steps of the approximate $\beta_1^{eq'}$, having all steps of equal size, in Figure 13.10a).

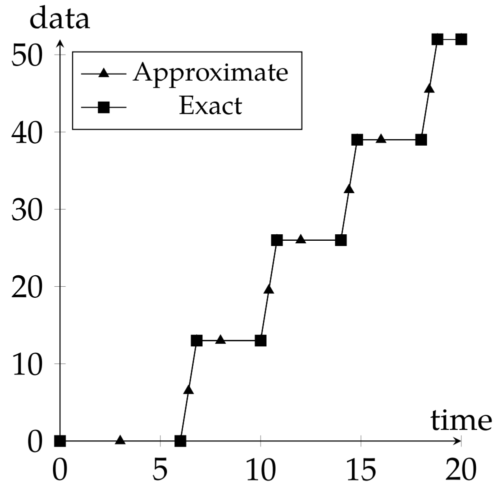
However, despite this, Figure 13.10b shows that this difference is irrelevant when computing the equivalent service curve for the whole tandem. As we compute the convolution of all β_i^{eq} (respectively, $\beta_i^{eq'}$), we obtain in fact $\beta^{eq} = \beta^{eq'}$, i.e., there is no information given by the exact method that is not captured from the approximate one as well. A similar phenomenon was observed in all our experiments.

The above observations cast the approximate method in a new – and more favorable – light. They suggest that the approximate method is as accurate as the exact one in an end-to-end context, enabling one to compute the exact same worst-case bounds with only a fraction of the computational cost, by avoiding costly computations that have no end effect on these results.

We stress the importance of such a finding – were it formally proven – since one can always find cases where, despite our optimizations, the exact method will just be too costly to execute, while the approximate one remains feasible.



(a) Comparison of equivalent service curves at node 1, i.e., β_1^{eq} and $\beta_1^{eq'}$.



(b) Comparison of equivalent end-to-end service curves of the tandem, i.e., β^{eq} and $\beta^{eq'}$.

Figure 13.10: Comparison of the results of the exact and approximate method.

Chapter 14

Isospeed: Improving Algorithms for (min,+) and (max,+) Convolution by Exploiting their Isomorphism

The work described in this chapter is the result of the most recent collaboration with Paul Nikolaus and Giovanni Stea.
An article discussing the (min,+) convolution and an earlier version of the by-sequence heuristic has been presented at ECRTS 2023 [ZNS23a; ZNS23b]. We are also working on an extended article, mentioning the due changes for (max,+) convolution and the improved by-sequence heuristic discussed here, to be submitted for peer-review sooner rather than later.
This work is inspired by the results [PLSK11] – we wish to thank Steffen Bondorf for pointing out this paper to us, as well as Raul-Paul Epure for suggestions with respect to some proofs.

The optimizations discussed in Chapter 13 improve dramatically the computational cost of a single convolution, though, as we pointed out, they apply only to subadditive curves. In practice, many instances of DNC can incur in long convolutions involving UPP curves which are not subadditive. We mentioned an example in Section 6.2, where the convolution of IWRR per-flow service curves is not subadditive and potentially complex.

The work in this chapter has been inspired by the work in [PLSK11], which first observed this form of performance optimizations, although they were unable to discern its root cause. Here, we expand on said result, discussing those root causes and provide improved algorithms based on this.

This chapter is organized as follows. In Section 14.1, we discuss the observations of [PLSK11], provide an explanation for the phenomenon, and discuss the opportunity of further improvements – which require some extensions to the mathematical framework. In Section 14.2, we provide such extension, outlining an isomorphism property that applies also to restricted functions. In Sections 14.3 and 14.4, we use the extended mathematical framework to provide and evaluate improved algorithms for, respectively, (min,+) and (max,+) convolution.

14.1 Explaining the algorithmic improvements via isomorphism

First, we recall in Algorithms 6 and 8 the *standard* algorithms for (min,+) and (max,+) convolution of UPP functions, as we have described them so far. Henceforth, we will refer to this as *direct* approach.

Algorithm 6 Pseudocode for (min,+) convolution

```
    Input Functions $f$ and $g$.
    Return Their (min,+) convolution $f \otimes g$.
 1: Decompose the operands as $f = f_t^{\wedge} \wedge f_p^{\wedge}$ and $g = g_t^{\wedge} \wedge g_p^{\wedge}$
 2: Compute $h_{tt} := f_t^{\wedge} \otimes g_t^{\wedge}$, $h_{tp} := f_t^{\wedge} \otimes g_p^{\wedge}$, $h_{pt} := f_p^{\wedge} \otimes g_t^{\wedge}$ as described in Section 3.4
 3: Compute $h_{pp} := f_p^{\wedge} \otimes g_p^{\wedge}$ as follows:
 4:     Compute $d = \operatorname{lcm}(d_f, d_g)$
 5:     Compute $c = d \cdot \min(\rho_f, \rho_g)$
 6:     Compute $T = T_f + T_g + d$
 7:     Compute
            $I_{f_p^{\wedge}} = [T_f, T_f + 2 \cdot d[,$
            $I_{g_p^{\wedge}} = [T_g, T_g + 2 \cdot d[,$
            $I_{\otimes_{pp}} = [T_f + T_g, T_f + T_g + 2 \cdot d[$
 8:     Compute $S_{\otimes_{pp}}^{I_{\otimes_{pp}}} = S_{f_p^{\wedge}}^{I_{f_p^{\wedge}}} \otimes S_{g_p^{\wedge}}^{I_{g_p^{\wedge}}}$ using Algorithm 7
 9:     $R_{h_{pp}} = \left(S_{\otimes_{pp}}^{I_{\otimes_{pp}}}, T, d, c\right)$
10: $f \otimes g = h_{tt} \wedge h_{tp} \wedge h_{pt} \wedge h_{pp}$
```

As we mentioned in Section 4.4, it is possible to replace the (min,+) convolution of left-continuous non-decreasing functions with a (max,+) convolution, by means of pseudoinverses.

Corollary 4.17 (Alternative Expression for (min,+) Convolution, via its Isomorphism). *Let f and g be functions of $\mathcal{U}$ that are left-continuous and non-decreasing. Then,*

$$f \otimes g \overset{(4.6)}{=} \left((f \otimes g)_{\uparrow}^{-1}\right)_{\downarrow}^{-1} \overset{(4.12)}{=} \left(f_{\uparrow}^{-1} \overline{\otimes} g_{\uparrow}^{-1}\right)_{\downarrow}^{-1}. \tag{4.13}$$

Algorithm 7 Pseudocode for *by-sequence* (min,+) convolution

```
    Input Sequences $S_f$ and $S_g$, and interval $I_\otimes$.
    Return The (min,+) convolution $S_\otimes^{I_\otimes} = S_f \otimes S_g$.
 1: Initialize $E$ as an empty set
 2: for all $e_f \in S_f, e_g \in S_f$ do
 3:     $E \leftarrow e_f \otimes e_g$
 4: end for
 5: Compute $S$ as the lower envelope of the set $E$
 6: Compute $S_\otimes^{I_\otimes}$ as the restriction of $S$ to $I_\otimes$
```

Algorithm 8 Pseudocode for (max,+) convolution

```
    Input Functions $f$ and $g$.
    Return Their (max,+) convolution $f \overline{\otimes} g$.
 1: Decompose the operands as $f = f_t^\vee \vee f_p^\vee$ and $g = g_t^\vee \vee g_p^\vee$
 2: Compute $h_{tt} := f_t^\vee \overline{\otimes} g_t^\vee$, $h_{tp} := f_t^\vee \overline{\otimes} g_p^\vee$, $h_{pt} := f_p^\vee \overline{\otimes} g_t^\vee$ as described in Section 3.7
 3: Compute $h_{pp} := f_p^\vee \overline{\otimes} g_p^\vee$ as follows:
 4:     Compute $d = \operatorname{lcm}(d_f, d_g)$
 5:     Compute $c = d \cdot \max(\rho_f, \rho_g)$
 6:     Compute $T = T_f + T_g + d$
 7:     Compute
            $I_{f_p^\vee} = [T_f, T_f + 2 \cdot d[,$
            $I_{g_p^\vee} = [T_g, T_g + 2 \cdot d[,$
            $I_{\overline{\otimes}_{pp}} = [T_f + T_g, T_f + T_g + 2 \cdot d[$
 8:     Compute $S_{\overline{\otimes}_{pp}}^{I_{\overline{\otimes}_{pp}}} = S_{f_p^\vee}^{I_{f_p^\vee}} \overline{\otimes} S_{g_p^\vee}^{I_{g_p^\vee}}$ using Algorithm 9
 9:     $R_{h_{pp}} = \left(S_{\overline{\otimes}_{pp}}^{I_{\overline{\otimes}_{pp}}}, T, d, c\right)$
10: $f \overline{\otimes} g = h_{tt} \vee h_{tp} \vee h_{pt} \vee h_{pp}$
```

Note that Algorithms 6 and 8 share the same algorithmic structure and complexity. In both, the most impactful operation is the by-sequence convolution (Line 8 of Algorithm 6 and Line 8 of Algorithm 8), whose complexity is superquadratic with the size of the cuts, i.e,

$$\mathcal{O}\left(n\left(S_{f_p^\wedge}^{I_{f_p^\wedge}}\right) \cdot n\left(S_{g_p^\wedge}^{I_{g_p^\wedge}}\right) \cdot \log\left(n\left(S_{f_p^\wedge}^{I_{f_p^\wedge}}\right) \cdot n\left(S_{g_p^\wedge}^{I_{g_p^\wedge}}\right)\right)\right),$$

$$\mathcal{O}\left(n\left(S_{f_p^\vee}^{I_{f_p^\vee}}\right) \cdot n\left(S_{g_p^\vee}^{I_{g_p^\vee}}\right) \cdot \log\left(n\left(S_{f_p^\vee}^{I_{f_p^\vee}}\right) \cdot n\left(S_{g_p^\vee}^{I_{g_p^\vee}}\right)\right)\right).$$

Algorithm 9 Pseudocode for *by-sequence* (max,+) convolution

```
Input Sequences $S_f$ and $S_g$, and interval $I_{\overline{\otimes}}$.
Return The (max,+) convolution $S_{\overline{\otimes}}^{I_{\overline{\otimes}}} = S_f \overline{\otimes} S_g$.
1: Initialize $E$ as an empty set
2: for all $e_f \in S_f, e_g \in S_f$ do
3:     $E \leftarrow e_f \overline{\otimes} e_g$
4: end for
5: Compute $S$ as the upper envelope of the set $E$
6: Compute $S_{\overline{\otimes}}^{I_{\overline{\otimes}}}$ as the restriction of $S$ to $I_{\overline{\otimes}}$
```

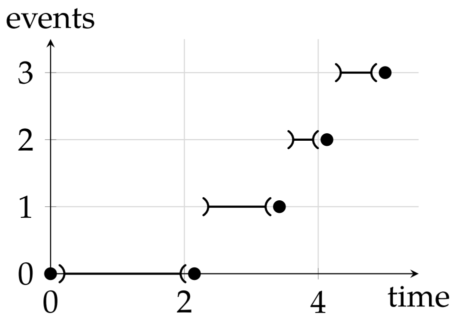


Figure 14.1: Example of event-based service curve.

In both, such sizes depend on $\mathrm{lcm}(d_f, d_g)$, meaning they can vary considerably depending on numerical properties of the operands.

Due to these similarities and being the pseudoinverses, albeit cheap, not free to execute, one would intuitively expect that applying Equation (4.13) (which we will, henceforth, refer to as *inverse* approach) would incur in longer computations. However, [PLSK11] observed that it is not the case, rather the computation time can be reduced this way by orders of magnitude. This work was based off the *RTC Toolbox*, and due to parts of the algorithm being written outside this tool and its closed-source nature, the authors were not able to discern whether the improvement was due to algebraic properties or inefficiencies in the implementation of *RTC Toolbox*.

We provide here an explanation that supports the first hypothesis. The use case addressed by the authors of [PLSK11] is the (min,+) convolution of a series of event-based service curves. Such curves, as exemplified in Figure 14.1, have time in $\mathbb{Q}_+$ on the x-axis and on the y-axis the number of events served in $\mathbb{N}_0$.

In the (min,+) convolution the dominant factor for the computational cost, as mentioned, is the size of the cuts $S_{f_p^\wedge}$ and $S_{f_p^\wedge}$, which depend in turn on $d_{f\otimes g} = \mathrm{lcm}(d_f, d_g)$, hence the numerical properties of d_f and d_g. In fact, let $\mathrm{lcm}(d_f, d_g) = k_{d_f} \cdot d_f = k_{d_g} \cdot d_g$, then $S_{f_p^\wedge}$ contains $2 \cdot k_{d_f}$ pseudo-periods of f, and $S_{g_p^\wedge}$ contains

$2 \cdot k_{d_g}$ pseudo-periods of g. We call k_{d_f} and k_{d_g} *extensions multipliers*.[1]

However, we show that using Equation (4.13) the computational cost depends instead on the numerical properties of c_f and c_g, which may be more favorable.

Proposition 14.1. *Let f and g be left-continuous, non-decreasing UPP functions. Then,*

$$d_{f \otimes g} = \max\left(\frac{d_f}{c_f}, \frac{d_g}{c_g}\right) \cdot \operatorname{lcm}\left(c_f, c_g\right), \tag{14.1}$$

$$c_{f \otimes g} = \operatorname{lcm}\left(c_f, c_g\right) \tag{14.2}$$

are sufficient period length and height for $f \otimes g$.

Proof. Since f and g are left-continuous, it holds by Equation (4.13) that

$$f \otimes g = \left(f_\uparrow^{-1} \overline{\otimes} g_\uparrow^{-1}\right)_\downarrow^{-1}.$$

Combining this with Proposition 3.30, we obtain for the inner function $f_\uparrow^{-1} \overline{\otimes} g_\uparrow^{-1}$

$$d_{f_\uparrow^{-1} \overline{\otimes} g_\uparrow^{-1}} = \operatorname{lcm}\left(d_{f_\uparrow^{-1}}, d_{g_\uparrow^{-1}}\right),$$

$$c_{f_\uparrow^{-1} \overline{\otimes} g_\uparrow^{-1}} = \max\left(\frac{c_{f_\uparrow^{-1}}}{d_{f_\uparrow^{-1}}}, \frac{c_{g_\uparrow^{-1}}}{d_{g_\uparrow^{-1}}}\right) \cdot d_{f_\uparrow^{-1} \overline{\otimes} g_\uparrow^{-1}}.$$

Using Theorem 10.4, we obtain for the period-length and height of the upper pseudoinverse

$$d_{f_\uparrow^{-1} \overline{\otimes} g_\uparrow^{-1}} = \operatorname{lcm}\left(c_f, c_g\right),$$

$$c_{f_\uparrow^{-1} \overline{\otimes} g_\uparrow^{-1}} = \max\left(\frac{d_f}{c_f}, \frac{d_g}{c_g}\right) \cdot \operatorname{lcm}\left(c_f, c_g\right).$$

Combining this with the outer function $(\cdot)_\downarrow^{-1}$, due to Theorem 10.2, we eventually obtain for $\left(f_\uparrow^{-1} \overline{\otimes} g_\uparrow^{-1}\right)_\downarrow^{-1}$ that

$$d_{f \otimes g} \overset{(4.13)}{=} d_{\left(f_\uparrow^{-1} \overline{\otimes} g_\uparrow^{-1}\right)_\downarrow^{-1}} = \max\left(\frac{d_f}{c_f}, \frac{d_g}{c_g}\right) \cdot \operatorname{lcm}\left(c_f, c_g\right),$$

$$c_{f \otimes g} \overset{(4.13)}{=} c_{\left(f_\uparrow^{-1} \overline{\otimes} g_\uparrow^{-1}\right)_\downarrow^{-1}} = \operatorname{lcm}\left(c_f, c_g\right).$$

This finishes the proof. □

[1] By definition of lcm, $k_{d_f}, k_{d_g} \in \mathbb{N}$.

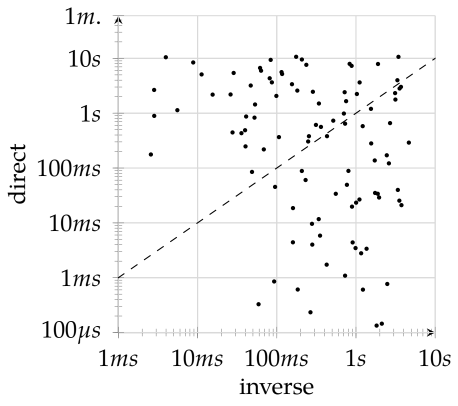


Figure 14.2: Extract from the following performance evaluation: (4.13) does not always improve runtime.

Note that this theorem gives us an alternative solution to the period length compared to the state of the art in Proposition 3.17. In this new expression, we see that the period length of the result is related to $\mathrm{lcm}(c_f, c_g)$, instead of $\mathrm{lcm}(d_f, d_g)$. This can provide vastly different performance, if the latter $\mathrm{lcm}(\cdot)$ is significantly closer to its operands than $\mathrm{lcm}(d_f, d_g)$, i.e., if the extension multipliers k_{c_f} and k_{c_g} are smaller than, respectively, k_{d_f} and k_{d_g}. This is likely the case in [PLSK11]: as we mentioned, their event-based service curve have $c_f, c_g \in \mathbb{N}_0$ and $d_f, d_g \in \mathbb{Q}_+$, which justifies the improved performance the authors observe when applying the transformation of Equation (4.13).

However, this may not be generally the case. For example, if one construct curves having breakpoints in $\mathbb{N}_0$ on the x-axis and in $\mathbb{Q}_+$ on the y-axis, using Equation (4.13) would, in most cases, actually result in degraded performance. This is exemplified in Figure 14.2, which is an extract of the following performance evaluation, comparing the two approaches for curves with randomized parameters.

Furthermore, we note that a similar isomorphism can be exploited for (max,+) convolutions.

Corollary 4.19 (Alternative Expression for (max,+) Convolution, via its Isomorphism)**.** *Let f and g be functions of $\mathcal{U}$ that are right-continuous and non-decreasing. Then,*

$$f \overline{\otimes} g \overset{(4.7)}{=} \left((f \overline{\otimes} g)^{-1}_{\downarrow}\right)^{-1}_{\uparrow} \overset{(4.14)}{=} \left(f^{-1}_{\downarrow} \otimes g^{-1}_{\downarrow}\right)^{-1}_{\uparrow}. \tag{4.15}$$

Thus, we can envisage a similar improvement for (max,+) convolution, which we show providing the analogous property of Proposition 14.1.

Proposition 14.2. *Let f and $g \in \mathcal{U}$ which are neither UC nor UI, and are right-continuous and non-decreasing. Then,*

$$d_{f \overline{\otimes} g} = \min\left(\frac{d_f}{c_f}, \frac{d_g}{c_g}\right) \cdot \operatorname{lcm}(c_f, c_g), \tag{14.3}$$

$$c_{f \overline{\otimes} g} = \operatorname{lcm}(c_f, c_g) \tag{14.4}$$

are sufficient period length and height for $f \overline{\otimes} g$.

A proof is provided in Appendix F.4.

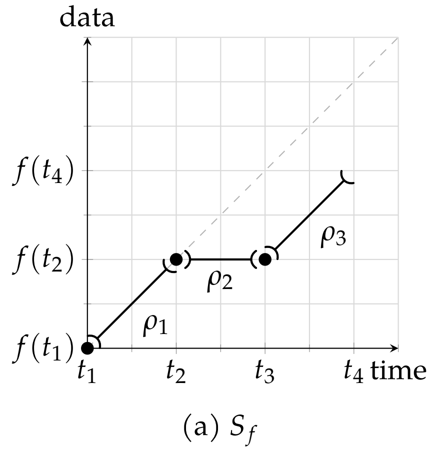
These properties suggest that we can predict which method (*direct* or *inverse*) is going to be more performant based on a comparison between $\operatorname{lcm}(d_f, d_g)$ and $\operatorname{lcm}(c_f, c_g)$. However, this comparison would not always appear as clear-cut. Let for example $d_f = 1$, $d_g = 7$, $c_f = 3$, $c_g = 2$. Then, we can compute that extension multipliers are $k_{d_f} = 7$, $k_{d_g} = 1$, $k_{c_f} = 2$, $k_{c_g} = 3$. Thus, we can expect to either a) use the *direct* approach, which will extend f by 14 pseudo-periods and g by only 2, or b) use the *inverse* one, which will extend f by 4 pseudo-periods and g by 6.

It is not a simple task, then, to compare the two choices and pick the better one, without inspecting the shapes of f and g. Moreover, as Sections 3.4 and 3.7 highlight, we cannot predict directly how a convolution is going to turn out in its entirety – we can only do so for the parts of its decomposition.

It is then on this decomposition that we focus in the following sections, as we can derive an isomorphism property that applies to $f_p^\wedge \otimes g_p^\wedge$ (and, for the case of the (max,+) convolution, $f_p^\vee \overline{\otimes} g_p^\vee$) enabling us to provide better insights on the overall runtime. Moreover, we use this to outline a new algorithm, which we call *isospeed*, that provides us the best parameters from both the *direct* and *inverse* approaches, which leads to better performance than both without requiring to actually perform the pseudoinverses.

14.1.1 Note on the *by-sequence* convolution

While the discussion above addresses only the *by-curve* algorithm and the extension multipliers, we note that also the *by-sequence* one can be affected by the inversion. We recall, as is discussed in Chapter 10, that the cardinality of a pseudoinverse can differ from that of the operand, due to plateaus and discontinuities. We show this with the example in Figure 14.3, where we can see that, even if the same "view" of f is used in both instances ($[t_1, t_4[$ on one axis, $[f(t_1), f(t_4)[$ on the other), the sequence in Figure 14.3b ends up having less *elements*, since plateaus "map" to a height difference, which is immaterial w.r.t. the computational cost of a *by-sequence* convolution.



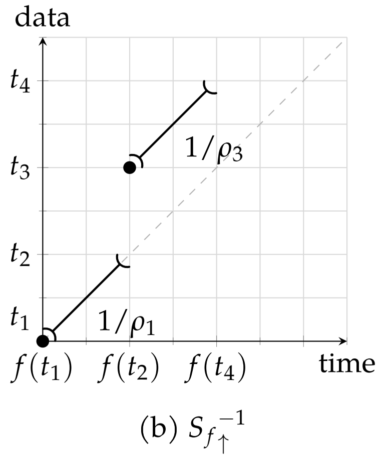


Figure 14.3: Example of upper pseudoinverse of a sequence S_f.

This can reflect also on the runtime of convolutions. In fact, since the *inverse* approach, in this example, would result in a shorter runtime compared to the *direct* approach.

Hence, even if one optimizes the number of extensions in the *by-curve* algorithm – as we will describe in the following sections – it may still be advantageous to perform the *by-sequence* convolution via the *inverse* approach in order to reduce the number of elements to be processed.

On the other hand, the number of elements is only *one* factor for the effective runtime of the by-sequence convolution. As is discussed in depth in Section 8.8, the runtime is also affected by topological properties that are difficult to understand *ex ante*. Thus, later in Section 14.3.1 we only provide a *heuristic* approach to this choice, that picks the *direct* or *inverse* by-sequence convolution based on the number of plateaus and jumps.

In the performance evaluation sections of this chapter, we will compare shapes of curves that are advantageous for this heuristic, i.e., with clearly more jumps than plateaus, or vice versa, and disadvantageous ones, i.e., with unclear tradeoffs between the two approaches. As these comparisons show, our approach still offers a significant improvement in the majority of cases.

14.2 Isomorphism for restricted functions

In the main results of this chapter, we exploit properties analogue to the isomorphisms proved in [Lie17], such as Corollary 4.17, which are applied however to functions restricted over a support, e.g., $f_p^\wedge$. The first obstacle is that the pseudoinverses are defined for non-decreasing functions, while the functions we consider are not. Thus, in this section we provide a generalization of the concept of pseudoinverses.

Definition 14.3 (Non-decreasing over Support). Let $f \in \mathcal{U}$, and let $D \subseteq \mathbb{Q}_+$. We say f is non-decreasing over D if and only if $f(s) \leq f(t)$ for any $s, t \in D$ such that $s \leq t$.

Naturally, if $f \in \mathcal{U}$ is non-decreasing, $f_I^\wedge$ is non-decreasing over I.

Definition 14.4 (Left-continuous over Support). Let $f \in \mathcal{U}$ and $D \subseteq \mathbb{Q}_+$. We say that f is left-continuous over D if, for any at $t_0 \in D$ such that exists $\delta_0 > 0$ so that $]t_0 - \delta_0, t_0] \subseteq D$, and for any $\varepsilon > 0$, there exists some $0 < \delta < \delta_0$ such that for all t with $t_0 - \delta < t < t_0$, it holds that

$$|f(t) - f(t_0)| < \varepsilon.$$

We also write $\lim_{t \nearrow t_0} f(t) = f(t_0)$.

For example, consider a function $f \in \mathcal{U}$ that is left-continuous and finite for all t and UPP from $T_f > 0$. Then, $f_p^\wedge$ will not be left-continuous, since $f_p^\wedge(t) = +\infty$ for all $t < T_f$ and $f_p^\wedge(T_f) < +\infty$. It will, however, be left-continuous over $\left[T_f, +\infty\right[$.

Definition 14.5 (Right-continuous over Support). Let $f \in \mathcal{U}$ and $D \subseteq \mathbb{Q}_+$. We say that f is right-continuous over D if, for any at $t_0 \in D$ such that exists $\delta_0 > 0$ so that $[[\, t_0, t_0 + \delta_0 \subseteq D$, and for any $\varepsilon > 0$, there exists some $0 < \delta < \delta_0$ such that for all t with $t_0 < t < t_0 + \delta$, it holds that

$$|f(t) - f(t_0)| < \varepsilon.$$

We also write $\lim_{t \searrow t_0} f(t) = f(t_0)$.

Naturally, if $f \in \mathcal{U}$ is right-continuous, $f_I^\wedge$ is right-continuous over I.

Definition 14.6 (Lower and Upper Pseudoinverse over an Interval). Let $f \in \mathcal{U}$ be non-decreasing over I, where $I = [a, +\infty[\subset \mathbb{Q}$. Then, its *lower pseudoinverse over* (*the interval*) I is defined as

$$f_{\downarrow,I}^{-1}(y) := \begin{cases} \inf\{t \in I \mid f(t) \geq y\}, & \text{if } y \geq f(a), \\ +\infty, & \text{otherwise,} \end{cases} \tag{14.5}$$

and its *upper pseudoinverse over* (*the interval*) I is defined as

$$f_{\uparrow,I}^{-1}(y) := \begin{cases} \sup\{t \in I \mid f(t) \leq y\}, & \text{if } y \geq f(a), \\ -\infty, & \text{otherwise.} \end{cases} \tag{14.6}$$

As discussed in Section 4.3, it does not hold in general that

$$\inf \{t \in I \mid f(t) \geq y\} = \sup \{t \in I \mid f(t) < y\},$$

for the lower pseudoinverse as well as

$$\sup \{t \in I \mid f(t) \leq y\} = \inf \{t \in I \mid f(t) > y\}$$

for the upper pseudoinverse. However, given $I = [a, +\infty[$, the two equations hold for $y > f(a)$ and $y \geq f(a)$, respectively. We state the above in the following proposition, whose proof can be derived by following the steps for Proposition 4.10 for a general $a \geq 0$ rather than 0.

Proposition 14.7. *Let $f \in \mathcal{U}$ be non-decreasing over interval $I = [a, +\infty[$. For all $y > f(a)$, its lower pseudoinverse is equal to*

$$f_{\downarrow}^{-1}(y) = \sup \{t \in I \mid f(t) < y\}, \tag{14.7}$$

and for all $y \geq f(a)$, its upper pseudoinverse is equal to

$$f_{\uparrow}^{-1}(y) = \inf \{t \in I \mid f(t) > y\}. \tag{14.8}$$

We also note that, since these pseudoinverses consider only values of $f(t)$ for $t \in I$, it follows that $f_{\downarrow,I}^{-1} = \left(f|_I^{\wedge}\right)_{\downarrow,I}^{-1} = \left(f|_I^{\vee}\right)_{\downarrow,I}^{-1}$, and similarly for $f_{\uparrow,I}^{-1}$.

Lemma 14.8. *Let $f \in \mathcal{U}$ be non-decreasing over I, where $I = [a, +\infty[\subset \mathbb{Q}$. Let $f(I) := [f(a), +\infty[$. Then, the lower pseudoinverse over I is left-continuous over $f(I)$, and the upper pseudoinverse over I is right-continuous over $f(I)$.*

The proof is easily derived from the one for Lemma 4.11, replacing 0 with a.

Theorem 14.9 (UPP properties of Lower Pseudoinverse over an Interval)**.** *Let I be an interval of the form $[a, +\infty[$. Let $f \in \mathcal{U}$ be neither UC nor UI, and non-decreasing over I. Then, its lower pseudoinverse over I, $f_{\downarrow,I}^{-1}$, is again a function of $\mathcal{U}$ with*

$$T_{f_{\downarrow,I}^{-1}} = \begin{cases} f\left(T_f + d_f\right), & \text{if } a \leq T_f, \\ f\left(a + d_f\right), & \text{if } a > T_f, \end{cases} \tag{14.9}$$

$$d_{f_{\downarrow,I}^{-1}} = c_f, \tag{14.10}$$

$$c_{f_{\downarrow,I}^{-1}} = d_f. \tag{14.11}$$

Theorem 14.10 (UPP properties of Upper Pseudoinverse over an Interval). *Let I be an interval of the form $[a, +\infty[$. Let $f \in \mathcal{U}$ be neither UC nor UI, and non-decreasing over I. Then, its upper pseudoinverse over I, $f^{-1}_{\uparrow,I}$, is again a function of $\mathcal{U}$ with*

$$T_{f^{-1}_{\uparrow,I}} = \begin{cases} f\left(T_f\right), & \text{if } a \leq T_f, \\ f\left(a\right), & \text{if } a > T_f, \end{cases} \tag{14.12}$$

$$d_{f^{-1}_{\uparrow,I}} = c_f \tag{14.13}$$

$$c_{f^{-1}_{\uparrow,I}} = d_f. \tag{14.14}$$

The proofs are easily derived following the steps of those of Theorems 10.2 and 10.4. The only difference is that, since we are considering values of $f(t)$ only for $t \in [a, +\infty[$, we can only consider $f(t)$ to be UPP from $\max\left(T_f, a\right)$. Note that, in the rest of this chapter, we will always use $a \leq T_f$, thus only the first branch of Equations (14.9) and (14.12) apply.

As briefly mentioned in Section 10.1, one can improve the result of Equation (14.9) using additional assumptions on the shape of f. As this will be useful in this chapter, we derive these results explicitly for pseudoinverses over interval.

Lemma 14.11 (Improved period start for Lower Pseudoinverse over Interval). *Let $f \in \mathcal{U}$ be neither UC nor UI, right-continuous, and non-decreasing over the interval $I = [a, +\infty[$, where $a \leq T_f$. Let*

$$\begin{aligned} T^*_f &:= f^{-1}_{\downarrow,I}(f(T_f)) \\ &= \inf\left\{t \geq a \mid f(t) \geq f(T_f)\right\} \\ &= \inf\left\{t \geq a \mid f(t) = f(T_f)\right\}, \end{aligned} \tag{14.15}$$

and

$$\begin{aligned} T^{**}_f &:= f^{-1}_{\downarrow,I}(f(T^*_1 + d_f)) \\ &= \inf\left\{t \geq a \mid f(t) = f(T^*_1 + d_f)\right\}. \end{aligned} \tag{14.16}$$

Then, if f is UPP from T^*_f and if $T^{**}_f = T^*_f + d_f$, the pseudo-periodic start $T_{f^{-1}_{\uparrow}}$ in Equation (14.9) can be improved into

$$T_{f^{-1}_{\downarrow,I}} = f(T_f). \tag{14.17}$$

A proof is provided in Appendix F.2. We note that, due to the way T^*_f and T^{**}_f are defined, the property can be verified with any period-start of f, be it *minimal* or not. The core issue in Lemma 14.11 is exemplified in Figure 14.4. In this example, the pseudo-period of f ends with a constant segment, with value 2. Hence, $f^{-1}_{\downarrow,p}(f(T_f + d_f)) = f^{-1}_{\downarrow,p}(2) = 4 < T_f + d_f$. As we can see in Figure 14.4b, this translates in a jump between (2,4) and (2,5), which results in $f^{-1}_{\downarrow,p}$ being UPP from $f(T_f + d_f) = 2$ rather than from $f(T_f) = 1$.

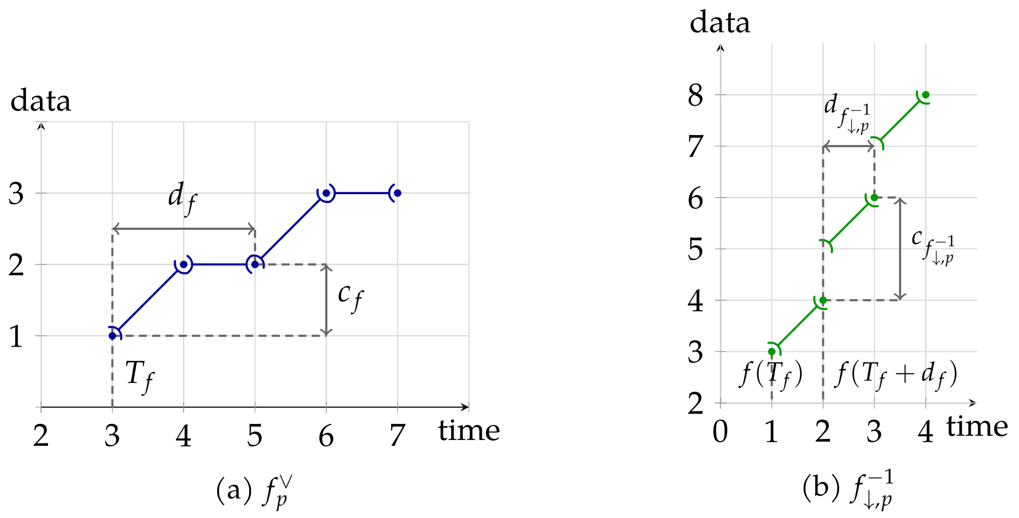


Figure 14.4: Example of lower pseudoinverse with $T_f^{**} < T_f + d_f$.

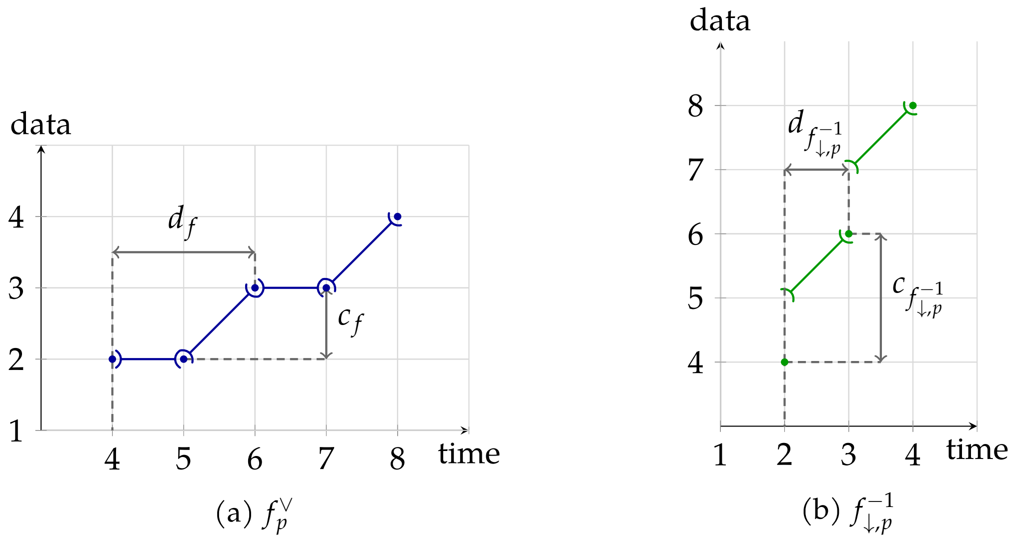


Figure 14.5: Example of lower pseudoinverse with $T_f^{**} = T_f + d_f$.

The opposite case can be seen in the example of Figure 14.5, where the constant segment is at the start rather than the end of the pseudo-period of f. In fact, $f^{-1}_{\downarrow,p}(f(T_f + d_f)) = f^{-1}_{\downarrow,p}(3) = 6 = T_f + d_f$. Thus, in Figure 14.5b we can see that the jump is present already at $f(T_f) = 2$, and the UPP property of $f^{-1}_{\downarrow,p}$ holds for $T_{f^{-1}_{\downarrow,p}} = f(T_f)$.

Remark 14.12. If the assumptions of Lemma 14.11 are not satisfied, we can alter the representation to make it so. In fact is trivial to verify that if we replace T_f with T_f^{**}, then Lemma 14.11 will always apply.

Lemma 14.13. *Let $f \in \mathcal{U}$ be neither UC nor UI, left-continuous, and non-decreasing over the interval $I = [a, +\infty[$, where $a \leq T_f$. Moreover, let $f^{-1}_{\uparrow,I}$ be its upper pseudoinverse over I. Then, $f^{-1}_{\uparrow,I}$ satisfies the conditions of Lemma 14.11. Thus, its lower pseudoinverse $\left(f^{-1}_{\uparrow,I}\right)^{-1}_{\downarrow,[f(a),+\infty[}$ is UPP from $f^{-1}_{\uparrow,I}(T_{f^{-1}_{\uparrow,I}}) = T_f$.*

A proof is provided in Appendix F.2.

Lemma 14.14 (Sufficient Cut for Lower Pseudoinverse over Interval)**.** *Let $f \in \mathcal{U}$ be neither UC nor UI, and is right-continuous and non-decreasing over $I = [a, +\infty[$. Then, in order to compute $f^{-1}_{\downarrow,I}(x)$ with $x \in [x_1, x_2] \subset [f(a), +\infty[$ and $x_1 < x_2$, it is sufficient to use $f(t)$ with $t \in [t_1, t_2]$, where $t_1 := f^{-1}_{\downarrow,I}(x_1)$ and $t_2 := f^{-1}_{\downarrow,I}(x_2)$.*

A proof is provided in Appendix F.2.

Lemma 14.15 (Sufficient Cut for Upper Pseudoinverse over Interval)**.** *Let $f \in \mathcal{U}$ be neither UC nor UI, and is left-continuous and non-decreasing over $I = [a, +\infty[$. Then, in order to compute $f^{-1}_{\uparrow,I}(x)$ and $x \in [x_1, x_2] \subset [f(a), +\infty[$ with $x_1 < x_2$, it is sufficient to use $f(t)$ with $t \in [t_1, t_2]$, where $t_1 := f^{-1}_{\uparrow,I}(x_1)$ and $t_2 := f^{-1}_{\uparrow,I}(x_2)$.*

A proof is provided in Appendix F.2. These results provide the necessary framework to derive similar results to Corollaries 4.17 and 4.19 for the convolution of periodic parts – which is the operation that we target to optimize. We derive these results in the following subsections, while in Sections 14.3 and 14.4 we use these results to explore their algebraic properties and derive improved algorithms.

14.2.1 Isomorphism of restricted (min,+) convolution

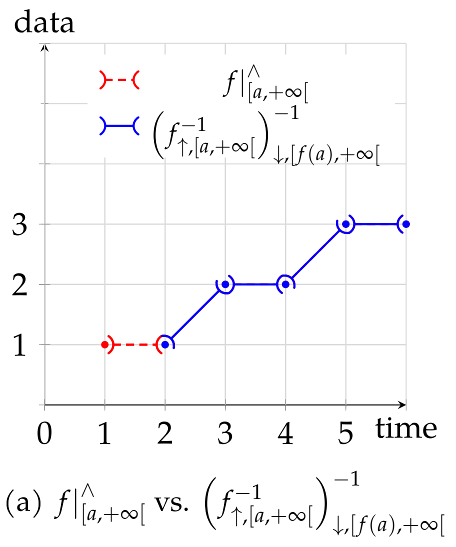
We provide a generalization of Theorem 4.16 for functions restricted to the pseudo-periodic part. Proofs for the following results are provided in Appendix F.2.1.

Lemma 14.16. *Let $f \in \mathcal{U}$ be non-decreasing over $I = [a, +\infty[\subset \mathbb{Q}_+$. Let $x \in I$. If $f(x) \leq y$, then $f^{-1}_{\uparrow,I}(y) \geq x$.*

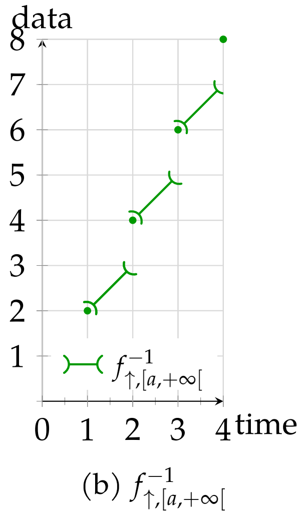
A proof is provided in Appendix F.2.1. Next, we generalize Proposition 4.1.

Proposition 14.17. *Let $f^{\wedge}_p, g^{\wedge}_p \in \mathcal{U}$ be left-continuous and non-decreasing, respectively, over $\left[T_f, +\infty\right[$ and $\left[T_g, +\infty\right[$. Then, for any $t \in \left[T_f + T_g, +\infty\right[$ it exists $s^* \in [T_f, t - T_g]$ such that*

$$\begin{aligned}(f^{\wedge}_p \otimes g^{\wedge}_p)(t) &= \inf_{0 \leq s \leq t} \left\{ f^{\wedge}_p(s) + g^{\wedge}_p(t-s) \right\} \\ &= \inf_{T_f \leq s \leq t - T_g} \left\{ f^{\wedge}_p(s) + g^{\wedge}_p(t-s) \right\} \\ &= f^{\wedge}_p(s^*) + g^{\wedge}_p(t - s^*).\end{aligned}$$



(a) $f|^{\wedge}_{[a,+\infty[}$ vs. $\left(f^{-1}_{\uparrow,[a,+\infty[}\right)^{-1}_{\downarrow,[f(a),+\infty[}$



(b) $f^{-1}_{\uparrow,[a,+\infty[}$

Figure 14.6: Example of loss of information when computing pseudoinverses over an interval.

In other words, the infimum is attainable.

A proof is provided in Appendix F.2.1.

Theorem 14.18. *Let $f_p^{\wedge}, g_p^{\wedge} \in \mathcal{U}$ be left-continuous and non-decreasing, respectively, over $[T_f, +\infty[$ and $[T_g, +\infty[$. Then*

$$\left(f_p^{\wedge} \otimes g_p^{\wedge}\right)^{-1}_{\uparrow,\left[T_f+T_g,+\infty\right[} = \left(f^{-1}_{\uparrow,p} \overline{\otimes} g^{-1}_{\uparrow,p}\right). \tag{14.18}$$

A proof is provided in Appendix F.2.1. Next, we also generalize Lemma 4.12.

Proposition 14.19. *Let $f \in \mathcal{U}$ be left-continuous and non-decreasing on the interval $I = [a, +\infty[$. Let $a' := \sup\{t \geq a \mid f(t) = f(a)\} \geq a$. Then*

$$\left(f^{-1}_{\uparrow,[a,+\infty[}\right)^{-1}_{\downarrow,[f(a),+\infty[} = f|^{\wedge}_{[a',+\infty[}.$$

A proof is provided in Appendix F.2.1. Note that Proposition 14.19 implies that performing the lower pseudoinverse (over an interval) of an upper pseudoinverse (over an interval) does not reconstitute f over that same interval, but only a subset of it: in fact, we obtain $f|^{\wedge}_{[a',+\infty[}$ instead of $f|^{\wedge}_{[a,+\infty[}$, where $a' \geq a$. We exemplify how this information loss happens in Figure 14.6, where $a = 1$, $f(a) = 1$ and $a' = 2$. We can see that f (Figure 14.6a) has a constant segment in $[a, a'[$, which is then represented in $f^{-1}_{\uparrow,[a,+\infty[}$ (Figure 14.6b) as the point $(f(a), a')$. Since this point is also the left boundary of the finite part of $f^{-1}_{\uparrow,[a,+\infty[}$, we lose information that would help reconstitute this constant segment, i.e., that there should be a jump from $(f(a)^-, a)$ to $(f(a), a')$.

On the other hand, if the value a is known, it is easy to reconstitute $f|^{\wedge}_{[a,+\infty[}$ by observing that the missing values of f in $[a, a'[$ are all $f(a')$, which is part of the result. Again referencing Figure 14.6a, we can see how the constant segment in red is the missing piece that can be reconstituted by knowing $a = 1$.

We formalize this process through the *reconstruction operator*, $[f]_a$. Given a function f that is either $+\infty$ or $-\infty$ in $[0, a'[$, and finite in $[a', +\infty[$, then

$$[f]_a(t) = \begin{cases} f(t), & \text{if } t \in [0, a[, \\ f(a'), & \text{if } t \in [a, a'[, \\ f(t), & \text{if } t \in [a', +\infty[. \end{cases} \tag{14.19}$$

With the help of the reconstruction operator, we can state a stronger version of Proposition 14.19.

Proposition 14.20. *Let $f \in \mathcal{U}$ be left-continuous and non-decreasing on the interval $I = [a, +\infty[$. Let $a' := \sup\{t \geq a \mid f(t) = f(a)\} \geq a$. Then*

$$\left[\left(f^{-1}_{\uparrow,[a,+\infty[}\right)^{-1}_{\downarrow,[f(a),+\infty[}\right]_a = f|^{\wedge}_{[a,+\infty[}. \tag{14.20}$$

A proof is provided in Appendix F.2.1. Generalizing Corollary 10.6, we can derive an improved start of the pseudo-periodic part under left-continuity. The proof is a direct consequence of Proposition 14.20.

Corollary 14.21. *Let f be a function of $\mathcal{U}$ which is left-continuous and non-decreasing. Define $I := [a, +\infty[\subset \mathbb{Q}$. Then,*

$$T_{\left[\left(f^{-1}_{\uparrow,[a,+\infty[}\right)^{-1}_{\downarrow,[f(a),+\infty[}\right]_a} = T_{f|^{\wedge}_{[a,+\infty[}}. \tag{14.21}$$

Combining these results, we can derive an alternative expression for the computation of $f^{\wedge}_p \otimes g^{\wedge}_p$, analogous to Corollary 4.17 for restricted functions.

Corollary 14.22 (Alternative Expression for (min,+) Convolution of Periodic Parts)**.** *Let $f^{\wedge}_p$ and $g^{\wedge}_p$ be functions of $\mathcal{U}$ that are left-continuous and non-decreasing over, respectively, $[T_f, +\infty[$ and $[T_g, +\infty[$. Then,*

$$f^{\wedge}_p \otimes g^{\wedge}_p = \left[\left(f^{-1}_{\uparrow,p} \overline{\otimes} g^{-1}_{\uparrow,p}\right)^{-1}_{\downarrow,\left[f(T_f)+g(T_g),+\infty\right[}\right]_{T_f+T_g}. \tag{14.22}$$

Corollary 14.22 is the critical result we will further explore in Section 14.3 to derive the algebraic properties driving our improved algorithm for the computation of $f^{\wedge}_p \otimes g^{\wedge}_p$. In the following subsection, we derive the analogous results for the (max,+) convolution.

14.2.2 Isomorphism of restricted (max,+) convolution

Similar properties can be derived for the (max,+) side, i.e., to derive an alternative expression for $f_p^\vee \overline{\otimes} g_p^\vee$. We thus provide a generalization of Theorem 4.18 for functions restricted to the pseudo-periodic part.

Proofs for the following results are provided in Appendix F.2.2.

Lemma 14.23. *Let $f \in \mathcal{U}$ be non-decreasing and $I = [a, +\infty[\subset \mathbb{Q}_+$. Let $x \in I$ and $y \geq f(a)$. If $f(x) \geq y$, then $f_{\downarrow,I}^{-1}(y) \leq x$.*

A proof is provided in Appendix F.2.2. The following lemma generalizes Proposition 4.3.

Proposition 14.24. *Let f and g be non-decreasing and right-continuous functions of $\mathcal{U}$. Then, for any $t \in \left[T_f + T_g, +\infty\right[$ it exists $s^* \in [T_f, t - T_g]$ such that*

$$\begin{aligned}
(f_p^\vee \overline{\otimes} g_p^\vee)(t) &= \sup_{0 \leq s \leq t} \left\{ f_p^\vee(s) + g_p^\vee(t-s) \right\} \\
&= \sup_{T_f \leq s \leq t - T_g} \left\{ f_p^\vee(s) + g_p^\vee(t-s) \right\} \\
&= f(s^*) + g(t - s^*)
\end{aligned}$$

In other words, the supremum is attainable.

A proof is provided in Appendix F.2.2.

Theorem 14.25. *Let f and g be right-continuous, non-decreasing UPP functions. Then*

$$\left(f_p^\vee \overline{\otimes} g_p^\vee \right)^{-1}_{\downarrow, \left[T_f + T_g, +\infty\right[} = \left(f_{\downarrow,p}^{-1} \otimes g_{\downarrow,p}^{-1} \right). \tag{14.23}$$

A proof is provided in Appendix F.2.2. Next, we also generalize Lemma 4.13.

Proposition 14.26. *Let $f \in \mathcal{U}$ be right-continuous and non-decreasing on the interval $I = [a, +\infty[$. Then*

$$\left(f_{\downarrow,[a,+\infty[}^{-1} \right)^{-1}_{\uparrow,[f(a),+\infty[} = f|_{[a,+\infty[}^\vee. \tag{14.24}$$

A proof is provided in Appendix F.2.2. Unlike Proposition 14.19, Proposition 14.26 does not imply any loss of information. In fact, even if there is a constant segment such as in Figure 14.6, the starting point a is preserved:

$$f_{\downarrow,[a,+\infty[}^{-1}(f(a)) = \inf\{t \geq a \mid f(t) \geq f(a)\} = a.$$

Generalizing Corollary 10.7, we can derive an improved start of the pseudo-periodic part under right-continuity. The proof is a direct consequence of Proposition 14.26.

Corollary 14.27. *Let $f : \mathbb{Q} \to \mathbb{Q} \cup \{+\infty, -\infty\}$ be a right-continuous, non-decreasing UPP function. Define $I := [a, +\infty[\subset \mathbb{Q}$. Then,*

$$T_{\left(f^{-1}_{\downarrow,[a,+\infty[}\right)^{-1}_{\uparrow,[f(a),+\infty[}} = T_{f|^{\vee}_{[a,+\infty[}}. \tag{14.25}$$

Combining these results, we can derive an alternative expression for the computation of $f_p^{\vee} \overline{\otimes} g_p^{\vee}$, analogous of Equation (4.15) for restricted functions.

Corollary 14.28 (Alternative Expression for (max,+) Convolution of Periodic Parts)**.** *Let $f_p^{\vee}$ and $g_p^{\vee}$ be functions of $\mathcal{U}$ that are right-continuous and non-decreasing over, respectively, $\left[T_f, +\infty\right[$ and $\left[T_g, +\infty\right[$. Then,*

$$f_p^{\vee} \overline{\otimes} g_p^{\vee} = \left(f^{-1}_{\downarrow,p} \otimes g^{-1}_{\downarrow,p}\right)^{-1}_{\downarrow,\left[f(T_f)+g(T_g),+\infty\right[} \tag{14.26}$$

Corollary 14.28 is the critical result we will further explore in Section 14.4 to derive the algebraic properties driving our improved algorithm for the computation of $f_p^{\vee} \otimes g_p^{\vee}$.

14.3 Exploiting the isomorphism to speed up the (min,+) convolution

Proposition 14.1 gives us an idea on how to exploit the isomorphism in the sense of deriving UPP properties for the (min,+) convolution through its (max,+) counterpart. However, the result cannot be directly applied in an algorithm, aside from actually doing the transformation as it was done in [PLSK11] – i.e., the *inverse* approach. Given the isomorphism of restricted functions we introduced in Section 14.2, we can now derive a similar result that enables us directly apply the optimization to the computation of $f_p^{\wedge} \otimes g_p^{\wedge}$. Moreover, we even show that a combination of *direct* together with the *inverse* approach is feasible. This combination, which we call *isospeed* approach, exploits the parameter improvements from both sides and is able to outperform both. Last, but not least, we introduce an algorithmic description of the entire improved calculation.

For the following theorems, we use in the following the shorthand notation $\otimes_{pp} := f_p^{\wedge} \otimes g_p^{\wedge}$, $\overline{\otimes}_p^{-1} := f^{-1}_{\uparrow,p} \overline{\otimes} g^{-1}_{\uparrow,p}$, and $f^{-1}_{\uparrow,p} := f^{-1}_{\uparrow,\left[T_f,+\infty\right[}$ whenever it clarifies the presentation.

Theorem 14.29. *Let $f, g \in \mathcal{U}$ be left-continuous and non-decreasing functions. Let*

$$k_{c_f} := \frac{\operatorname{lcm}(c_f, c_g)}{c_f}, \tag{14.27}$$

$$k_{c_g} := \frac{\operatorname{lcm}(c_f, c_g)}{c_g}. \tag{14.28}$$

Then, $f_p^\wedge \otimes g_p^\wedge$ is again $\in \mathcal{U}$ with

$$T_{\otimes_{pp}} = \sup\left\{t \geq T_f + T_g \mid f_p^\wedge \otimes g_p^\wedge(t) \leq f(T_f) + g(T_g) + \operatorname{lcm}\left(c_f, c_g\right)\right\}, \tag{14.29}$$

$$d_{\otimes_{pp}} = \max\left(\frac{d_f}{c_f}, \frac{d_g}{c_g}\right) \cdot \operatorname{lcm}\left(c_f, c_g\right) = \max\left(k_{c_g} \cdot d_g, k_{c_f} \cdot d_f\right), \tag{14.30}$$

$$c_{\otimes_{pp}} = \operatorname{lcm}\left(c_f, c_g\right). \tag{14.31}$$

A proof is provided in Appendix F.3. The theorem provides alternative UPP properties for $f_p^\wedge \otimes g_p^\wedge$ through the isomorphism that links this computation to $f_{\uparrow,p}^{-1} \overline{\otimes} g_{\uparrow,p}^{-1}$. For $d_{\otimes_{pp}}$ and $c_{\otimes_{pp}}$, we can therefore combine these results together with the *direct* approach to leverage from both. Let $d'_{\otimes_{pp}}$ be the result from Equation (3.9), and $d''_{\otimes_{pp}}$ be the result of Equation (14.30). Then we can use

$$d_{\otimes_{pp}} = \min\left(d'_{\otimes_{pp}}, d''_{\otimes_{pp}}\right), \tag{14.32}$$

$$c_{\otimes_{pp}} = d_{\otimes_{pp}} \cdot \min\left(\frac{c_f}{d_f}, \frac{c_g}{d_g}\right). \tag{14.33}$$

For $T_{\otimes_{pp}}$, however, we have that the alternative expression (Equation (14.29)) cannot be computed *a priori*, i.e., without performing the computation of $f_p^\wedge \otimes g_p^\wedge$ first. However, we can still leverage its insight to optimize the convolution as we compute it, since we can check the result and cut away any part of the *by-sequence* convolution with value over $f(T_f) + g(T_g) + 2 \cdot \operatorname{lcm}\left(c_f, c_g\right)$. Moreover, since for any *by-element* convolution $e_f \otimes e_g(t) \geq f \otimes g$, we can anticipate such filter to the by-element convolutions, omitting them from the lower envelope. We show how this is implemented, together with the rest of the optimization technique, later on in Section 14.3.1. The isomorphism can be further explored as in the following results, to discuss the *cuts* of f and g required for the computation.

Corollary 14.30. *Let f and $g \in \mathcal{U}$ which are neither UC nor UI, and are left-continuous and non-decreasing in $\left[T_f, +\infty\right[$ and $\left[T_g, +\infty\right[$, respectively. Let $k_{c_f} := \frac{\operatorname{lcm}\left(c_f, c_g\right)}{c_f}$, $k_{c_g} := \frac{\operatorname{lcm}\left(c_f, c_g\right)}{c_g}$. Then, to compute $f_p^\wedge \otimes g_p^\wedge$ via the (max,+) isomorphism (Equation (14.22)), it* *is sufficient to use* $S_{f_p^\wedge}^{I_{f_p^\wedge}}$ *and* $S_{g_p^\wedge}^{I_{g_p^\wedge}}$ *with*

$$\begin{aligned} I_{f_p^\wedge} &= \left[T_f, T'_f + 2 \cdot k_{c_f} \cdot d_f\right], \\ I_{g_p^\wedge} &= \left[T_g, T'_g + 2 \cdot k_{c_g} \cdot d_g\right]. \end{aligned} \tag{14.34}$$

where we used[2]

$$\begin{aligned} T'_f &= \sup\left\{t \geq T_f \mid f(t) = f(T_f)\right\}, \\ T'_g &= \sup\left\{t \geq T_g \mid g(t) = g(T_g)\right\}. \end{aligned}$$

A proof is provided in Appendix F.3. Corollary 14.30 states that, instead of computing $f_p^\wedge \otimes g_p^\wedge$ using the intervals $I_{f_p^\wedge}$ and $I_{g_p^\wedge}$ described in Proposition 3.16, by exploiting the isomorphism we can use instead the alternative expressions provided by Equation (14.34). As before, these intervals can be much smaller, and therefore much more efficient to do computations with, since their size depends on $\mathrm{lcm}\left(c_f, c_g\right)$ rather than $\mathrm{lcm}\left(d_f, d_g\right)$.

However, this link is proved by using the (max,+) convolution, thus the proof does not establish that we can use these intervals as a direct replacement within the (min,+) convolution. Moreover, replacing *both* intervals is something we could already do (through the *inverse* approach) while in Section 14.1 we have shown, through an example, that it may be the case where, e.g., $f_p^\wedge$ has a smaller interval using the *direct* approach and $g_p^\wedge$ through the *inverse* approach – we would thus prefer to be able to *mix and match* these intervals to minimize the cuts taken of both functions. We address these issues in the following theorem.

Theorem 14.31 (Mix and Match ((min,+) Convolution))**.** *Let f and $g \in \mathcal{U}$ which are neither UC nor UI, and are left-continuous and non-decreasing in $\left[T_f, +\infty\right[$ and $\left[T_g, +\infty\right[$, respectively. Let $I_{f_p^\wedge}, I_{g_p^\wedge}$ be the intervals sufficient to compute $f_p^\wedge \otimes g_p^\wedge$ according to Proposition 3.16, and let $I'_{f_p^\wedge}, I'_{g_p^\wedge}$ be the intervals sufficient to compute $f_{\uparrow,p}^{-1} \overline{\otimes} g_{\uparrow,p}^{-1}$ according to Corollary 14.30.*

Then $I_{f_p^\wedge} \cap I'_{f_p^\wedge}, I_{g_p^\wedge} \cap I'_{g_p^\wedge}$ are intervals sufficient to compute $f_p^\wedge \otimes g_p^\wedge$.

A proof is provided in Appendix F.3. The above theorem states that, given different cut intervals for $f_p^\wedge$ and $g_p^\wedge$ computed with the two methods, we can pick the shortest of each pair for our computations. As we recall that the *by-sequence* convolution complexity is (Section 3.4.2)

$$\mathcal{O}\left(n\left(S_{f_p^\wedge}\right) \cdot n\left(S_{g_p^\wedge}\right) \cdot \log\left(n\left(S_{f_p^\wedge}\right) \cdot n\left(S_{g_p^\wedge}\right)\right)\right),$$

[2]The suprema are attainable since the functions are left-continuous over the respective intervals.

being able to use $I_{f_p^\wedge}, I'_{g_p^\wedge}$ allows us then to outperform both methods. This is confirmed empirically in Section 14.3.2, where we show the performance improvements obtained.

14.3.1 Improved algorithm for (min,+) convolution

We now show, gathering the previous results, how we can improve the algorithm for the computation of the (min,+) convolution of left-continuous, non-decreasing functions. We note that, compared to the algorithm whose implementation we discussed in Section 8.6, we will only change its $f_p^\wedge \otimes g_p^\wedge$ component, whose simplified code is shown in Listing 8.9.

The results above provide two improvements: an improved $d_{\otimes_{pp}}$ and an improved $T_{\otimes_{pp}}$. Both can be used to reduce the cuts of the functions, $f_p^\wedge$ and $g_p^\wedge$, that are used during the *by-sequence* algorithm. However, the improved $T_{\otimes_{pp}}$ cannot be computed *a priori*, as we recall (Equation (14.29))

$$T_{\otimes_{pp}} = \sup \left\{ t \geq T_f + T_g \mid f_p^\wedge \otimes g_p^\wedge(t) \leq f(T_f) + g(T_g) + \operatorname{lcm}\left(c_f, c_g\right) \right\}.$$

We can use information, though, *during* the convolution. In fact, we can compute the *by-sequence* convolution (let it be S) and then compute $T_{\otimes_{pp}}$ as the minimum between $T_f + T_g + d_{\otimes_{pp}}$ (Equation (3.8)) and the earliest time S reaches $f(T_f) + g(T_g) + \operatorname{lcm}\left(c_f, c_g\right)$. Furthermore, for this computation we can use using the optimized cuts provided by Theorem 14.31.

The resulting algorithm is shown in Algorithm 10, which emphasizes that we can improve the cuts of both the operands and the result using the improved parameters.

As mentioned, we can further improve this algorithm by anticipating the removal of the tail end of $S_{\otimes_{pp}}^{I_{\otimes_{pp}}}$ as a filter in the by-element convolutions, omitting them from the lower envelope. Recalling the implementation of the *by-sequence* convolution in Listing 8.11, we extend it as shown in Listing 14.1.[3] We can therefore use the insights from both the *direct* and *inverse* approaches to filter the convolutions "horizontally", with $T_f + T_g + 2 \cdot d$ as `cutEnd`, and "vertically", with $f(T_f) + g(T_g) + 2 \cdot \operatorname{lcm}\left(c_f, c_g\right)$ as `cutCeiling`. Note that this can be done safely since the convolution computes the *lower* envelope, hence the convolutions that are removed this way do not affect its result in any way.

The last phenomenon to address is then the effect of plateaus and discontinuities under pseudoinversion. Using the results above, we end up with optimal cuts S_f and S_g. Intuitively, one would expected that computing $\left(S_{f_\uparrow}^{-1} \overline{\otimes} S_{g_\uparrow}^{-1}\right)_\downarrow^{-1}$ (*inverse* by-sequence (min,+) convolution) rather than $S_f \otimes S_g$ (*direct* by-sequence (min,+)

[3]We are omitting, for simplicity, the other improvements discussed in Section 8.7, such as parallelization and partitioning.

Algorithm 10 Improved pseudocode for (min,+) convolution of periodic parts

```
Input Functions $f_p^\wedge$ and $g_p^\wedge$, which are both left-continuous and non-decreasing over their respective support.
Return Their (min,+) convolution $f_p^\wedge \otimes g_p^\wedge$.
1:  if $f_p^\wedge, g_p^\wedge$ are not left-continuous and non-decreasing then
2:      Continue with the usual algorithm (see Algorithm 6)
3:  else
4:      Compute $k_{c_f} \overset{(14.27)}{=} \frac{\operatorname{lcm}(c_f, c_g)}{c_f}$ and $k_{c_g} \overset{(14.28)}{=} \frac{\operatorname{lcm}(c_f, c_g)}{c_g}$
5:      Compute $d = \min\left(\operatorname{lcm}(d_f, d_g), \max\left(k_{c_f} d_f, k_{c_g} d_g\right)\right)$
6:      Compute $c = d \cdot \min\left(\frac{c_f}{d_f}, \frac{c_g}{d_g}\right)$
7:      Compute $T_1 = T_f + T_g + d$
            $I_{f_p} = \left[T_f, T_f + \min\left(\operatorname{lcm}(d_f, d_g), k_{c_f} d_f\right)\right[,$
            $I_{g_p} = \left[T_g, T_g + \min\left(\operatorname{lcm}(d_f, d_g), k_{c_g} d_g\right)\right[$
8:      Compute $S_{\otimes_{pp}}^{I_{\otimes pp}} = S_{f_p}^{I_{f_p}} \otimes S_{g_p}^{I_{g_p}}$
9:      Compute $T_2 = \sup\left\{t \geq T_f + T_g \mid S_{\otimes_{pp}}^{I_{\otimes pp}}(t) \leq f(T_f) + g(T_g) + \operatorname{lcm}(c_f, c_g)\right\}$
10:     Compute $T = \min(T_1, T_2)$
11:     Remove from $S_{\otimes_{pp}}^{\otimes pp}$ elements whose support is after $T + d$
12:     $h_{pp} := f_p \otimes g_p = \left(S_{\otimes_{pp}}^{\otimes pp}, T, d, c\right)$
13: end if
```

convolution) would be inefficient. However, as mentioned at the start of this chapter, pseudoinversion maps plateaus (which count as elements in the convolution runtime) into discontinuities (which do not), and vice versa. Thus, it may be the case that, given a by-sequence convolution applying the isomorphism or not to compute it has still a measurable impact on runtime.

On the other hand, this impact is difficult to accurately predict, since elements affect the runtime differently based on topological properties (see Section 8.8). Our *heuristic* algorithm estimates the computational cost of the two approaches by ignoring this last point, and considering instead each element in the convolution as having the same impact. We thus compute the number of elements involved as $C := n(S_f) \cdot n(S_g)$ and $D := n\left(S_{f_\uparrow^{-1}}\right) \cdot n\left(S_{g_\uparrow^{-1}}\right)$. Then, if $D < C$, we use the *inverse* approach, and the *direct* one if $C \leq D$. Note that we do not need to compute the pseudoinverses to perform this check: following the procedure described in Table 10.1, one can devise a simple (yet tedious) algorithm to derive both $n(S_f)$ and $n\left(S_{f_\uparrow^{-1}}\right)$ with a single linear scan of S_f. The resulting algorithm is sketched in

Listing 14.1 Scheme of a `Sequence.Convolution()` method that filters both *vertically* and *horizontally*

```
Sequence Convolution(
    Sequence sf, Sequence sg,
    Rational cutEnd = Rational.PlusInfinity,
    Rational cutCeiling = Rational.PlusInfinity)
{
    var convolutionPairs = GetFilteredElementPairs(sf, sg, cutEnd, cutCeiling);
    var convolutionElements = new List<Element>();
    foreach(var pair in convolutionPairs)
        convolutionElements.AddRange(Element.Convolution(pair.ef, pair.eg));
    var le = LowerEnvelope(convolutionElements);
    return le;
}

IEnumerable<(Element ef, Element eg)> GetFilteredElementPairs(
    Sequence sf, Sequence sg, Rational cutEnd, Rational cutCeiling)
{
    return GetElementPairs(sf, sg)
        .Where(pair => pair.ef.IsFinite && pair.eg.IsFinite)
        .Where(pair => pair.ef.StartTime + pair.eg.StartTime < cutEnd)
        .Where(pair => pair.ef.ValueAtStart + pair.eg.ValueAtStart <= cutCeiling);
}

IEnumerable<(Element ef, Element eg)> GetElementPairs(Sequence sf, Sequence sg) {
    foreach(var ef in sf)
        foreach(var eg in sg)
            yield return (ef, eg);
}
```

Algorithm 11.

The above heuristic has linear complexity, hence is quite inexpensive, and can be expected to identify the fastest way to perform a by-sequence convolution most of the times. However, it may not be accurate when operands have both plateaus and discontinuities, as we will highlight in the performance evaluation section.

These algorithmic components all attempt to reduce the number of unnecessary elementary convolutions computed, in order to improve runtime. We sketched here each individually, for the sake of conciseness. The interested reader can find their full integration, together with the other optimizations discussed in this thesis, in the source code of *Nancy* [ZSd].

Algorithm 11 Pseudocode for *by-sequence* (min,+) convolution, with heuristic

```
    Input Sequences $S_f$ and $S_g$.
    Return The (min,+) convolution $S_{\otimes}^{I_{\otimes}^{h}} = S_f \otimes S_g$.
1: Compute $n(S_f)$, $n\left((S_f)_{\uparrow}^{-1}\right)$, $n(S_g)$ and $n\left((S_g)_{\uparrow}^{-1}\right)$
2: $C \leftarrow n(S_f) \cdot n(S_g)$
3: $D \leftarrow n\left((S_f)_{\uparrow}^{-1}\right) \cdot n\left((S_g)_{\uparrow}^{-1}\right)$
4: if D > C then
5:     Use the standard algorithm, computing $S_f \otimes S_g$
6: else
7:     Use the by-sequence isomorphism, computing $\left((S_f)_{\uparrow}^{-1} \overline{\otimes} (S_g)_{\uparrow}^{-1}\right)_{\downarrow}^{-1}$
8: end if
```

14.3.2 Performance evaluation

In this subsection we show the performance improvements obtained from implementing these optimizations in *Nancy*.

First, we discuss the use case of a tandem of IWRR schedulers, mentioned in Section 11.4 and, in particular, Listing 11.3. As it is often the case, depending on the numerical properties of the operands a study like this, where we compute the (min,+) convolution of generic UPP curves, can become computationally demanding. It takes little tweaking of the parameters to find such case, as those in Listing 14.2. Moreover, as the service curves modelling IWRR nodes are not subadditive, the optimizations in Chapter 13 do not apply. Running this example with *Nancy* (on System 3) using the *direct* approach took 11 minutes and 12 seconds. However, using the *isospeed* approach the same computation took 0.8 seconds, a significant improvement that aligns with the first observations from [PLSK11]. However, as we mentioned before, there are instances where using the *inverse* approach as suggested in [PLSK11], would incur in worse performance than the *direct* approach. As we show, the *isospeed* approach instead uses the best combination of parameters from both, thus improving the parameters of the *by-curve* algorithm in all cases.

However, there are two phenomena that may counter-balance this statement. On one hand, we have that checking the hypotheses, computing the improved parameters, as well as performing the "vertical" filter in the *by-sequence* convolution, are an overhead over the standard algorithm that will appear as a loss of performance in those cases where they do not produce beneficial results. On the other hand, we have the phenomenon of plateaus and discontinuities mentioned in Section 14.1.1. It is to be expected that if we have many plateaus and few discontinuities, or vice versa, our heuristic will accurately pick which of the two approaches (*direct* or *inverse*) will produce a much more efficient *by-sequence* convolution, since discontinuities are im-

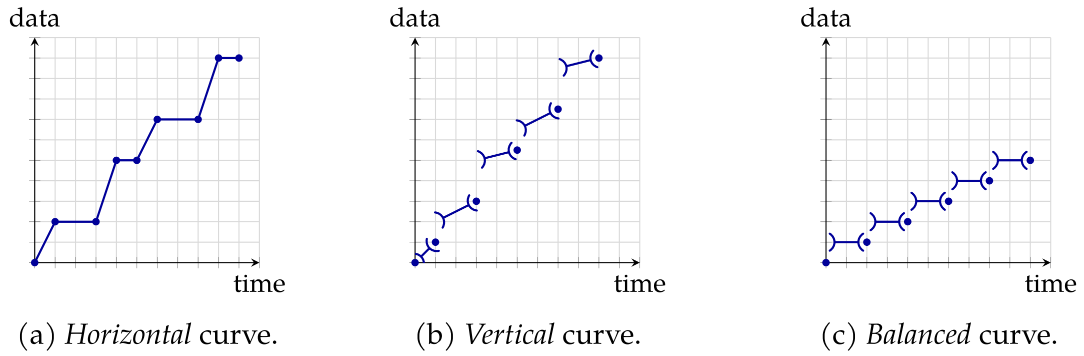
Listing 14.2 Parameters for a computationally demanding tandem of IWRR schedulers.

```
// ms as time unit, bit as data unit

// the flow has a max arrival rate of 10 Mbps,
// and is guaranteed such bandwidth at each node
var foi_ac = new SigmaRhoArrivalCurve(1024, 10000);

// 100 Mb/s, 1 ms of latency
// 3 cross flows
var b1 = new RateLatencyServiceCurve(100000, 1);
var weights_1 = new []{10, 30, 40, 20};
var l_min_1 = new []{512, 512, 1024, 1024};
var l_max_1 = new []{1024, 1024, 2048, 2048};

// 200 Mb/s, 1 ms of latency
// 5 cross flows
var b2 = new RateLatencyServiceCurve(200000, 1);
var weights_2 = new []{10, 20, 40, 50, 30, 50};
var l_min_2 = new []{1024, 1024, 1024, 1240, 1240, 1240};
var l_max_2 = new []{1240, 1240, 1240, 2480, 2480, 2480};

var b_eq_1 = IwrrServiceCurve(0, weights_1, l_min_1, l_max_1, b1);
var b_eq_2 = IwrrServiceCurve(0, weights_2, l_min_2, l_max_2, b2);

var b_eq = Curve.Convolution(b_eq_1, b_eq_2);   // we measure this step
```

material from the point of view of algorithmic complexity.

Hence, in the following performance tests, we will compare the performance of the three methods on three different kinds of curves, so that we can have a clearer view of these forces and the tradeoff in general. These three kinds of curves, depicted in Figure 14.7, provide different characteristics from the point of view of plateaus and discontinuities. We can expect that *horizontal* curves favor the *inverse* approach, while *vertical* favor the *direct* one. However, the distinction is clear enough that the heuristic will work in the majority cases, highlighting all the benefits of the *isospeed* algorithm. On the other hand, *balanced* curves will hard to predict for the heuristic, thus we expect to see that the optimization gains are in some instances cancelled out.

We run the experiments on System 2, using randomly generated parameters for the shapes discussed above. The results of our tests are shown in Figures 14.8 to 14.10. Note that, in each of this, the same experiments have been run in all three methods, and their results are being compared two approaches at a time. Thus, each point represents a set of parameters and the time took by one approach (its value over the x-axis) vs. the time it took with the other (its value over the y-axis). Then,



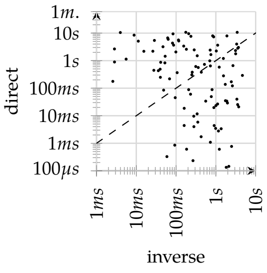
(a) *Horizontal* curve. (b) *Vertical* curve. (c) *Balanced* curve.

Figure 14.7: Shapes used for performance evaluation of (min,+) convolution.

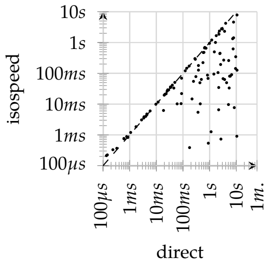
we also compare the *isospeed* algorithm against the best between *direct* and *inverse*. This last comparison highlights, on one hand, that our improvements successfully identify the reasons making one approach better than the other *a priori*, and on the other hand, that it can also beat both by avoiding trade-offs and taking the best parameters from each.

First, we notice that in many cases observed the *isospeed* approach introduces very significant improvements. From Figures 14.8a, 14.9a and 14.10a, we note that the experiments are generally well distributed in favor of either the *direct* or *inverse* approach – thus suggesting that blindly applying either of the two would not be a favorable choice. However, as the other figures show, the *isospeed* approach generally improves on both. Furthermore, we can notice that even in those cases where the *direct* approach was faster than the *inverse* one, *isospeed* still provided an improvement. This is thanks to Theorem 14.31, as we select the best cut for both operands independently.

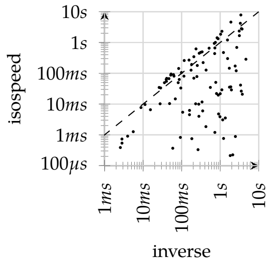
We observe that those cases where *isospeed* did not provide significant improvements over another approach form, in the figures, a line on or parallel to the bisector: from the distance of this line we can assess the overhead cost that the algorithm introduces, which is minimal. Finally, we note from Figure 14.8 that even when the heuristic is likely to fail, the *isospeed* algorithm can still provide runtime improvement that are generally much more impactful than the loss of performance in the unfavorable cases.



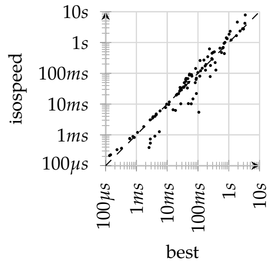
(a) *inverse* vs. *direct*.



(b) *direct* vs. *isospeed*.

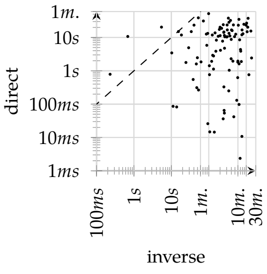


(c) *inverse* vs. *isospeed*.

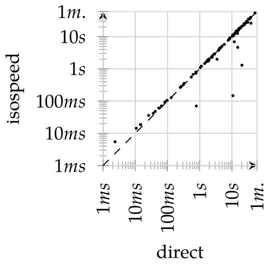


(d) *isospeed* vs. *best*.

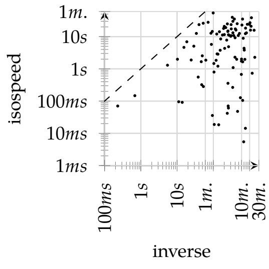
Figure 14.8: Performance comparison of the three algorithms for the (min,+) convolution of *balanced* curves.



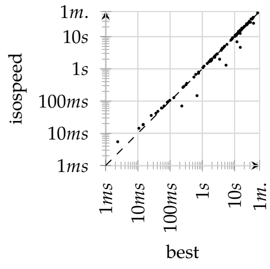
(a) *inverse* vs. *direct*.



(b) *direct* vs. *isospeed*.

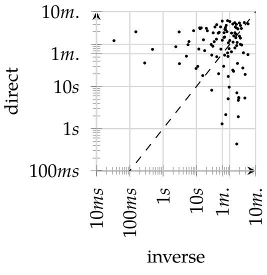


(c) *inverse* vs. *isospeed*.

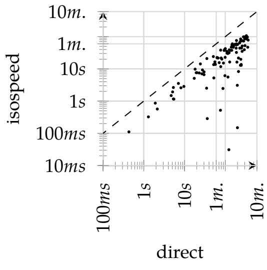


(d) *isospeed* vs. *best*.

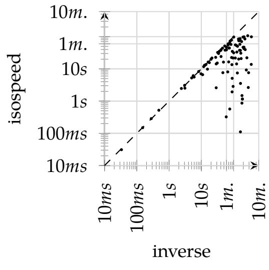
Figure 14.9: Performance comparison of the three algorithms for the (min,+) convolution of *vertical* curves.



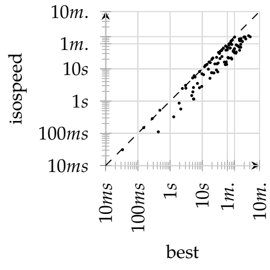
(a) *inverse* vs. *direct*.



(b) *direct* vs. *isospeed*.

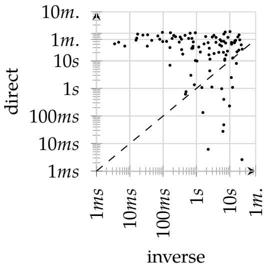


(c) *inverse* vs. *isospeed*.

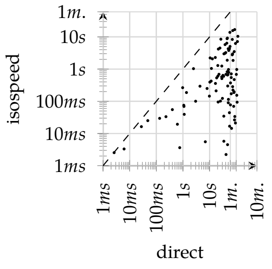


(d) *isospeed* vs. *best*.

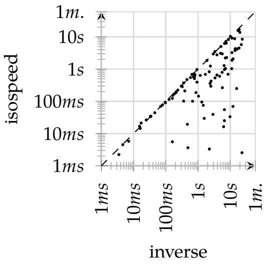
Figure 14.10: Performance comparison of the three algorithms for the (min,+) convolution of *horizontal* curves.



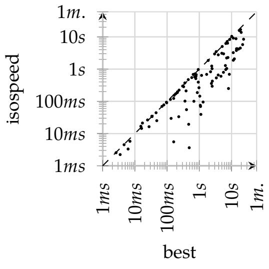
(a) *inverse* vs. *direct*.



(b) *direct* vs. *isospeed*.



(c) *inverse* vs. *isospeed*.



(d) *isospeed* vs. *best*.

Figure 14.11: Performance comparison of the three algorithms for (min,+) convolution. Operands are *horizontal* curves, and their parameters are set so that comparing extension multipliers is inconclusive.

14.4 Exploiting the isomorphism to speed up the (max,+) convolution

The insights used in the previous section can be exploited to prove an improved algorithm for (max,+) convolutions as well. Proposition 14.2 suggests how the computation may be improved, but does not give an algorithm aside from applying the transformation (i.e., the *inverse* approach).

For the following theorems, we will use the shorthand notation $\overline{\otimes}_{pp} := f_p^{\vee} \overline{\otimes} g_p^{\vee}$, $\otimes_p^{-1} := f_{\downarrow,p}^{-1} \otimes g_{\downarrow,p}^{-1}$, and $f_{\downarrow,p}^{-1} := f^{-1}_{\downarrow,\left[T_f,+\infty\right[}$ whenever it clarifies the presentation.

We will discuss functions that satisfy the condition of Lemma 14.11, i.e., such that $T_{f_{\downarrow,p}^{-1}} = f(T_f)$, $T_{g_{\downarrow,p}^{-1}} = g(T_g)$. When such assumption does not hold, we can use Remark 14.12 to obtain a new decomposition that satisfies Lemma 14.11. We will discuss how this can be done in practice in Section 14.4.1.

Theorem 14.32. *Let f and $g \in \mathcal{U}$ which are neither UC nor UI, and are right-continuous and non-decreasing. Moreover, let f and g satisfy the assumptions of Lemma 14.11, such that $T_{f_{\downarrow,p}^{-1}} = f(T_f)$, $T_{g_{\downarrow,p}^{-1}} = g(T_g)$. Let*

$$k_{c_f} := \frac{\operatorname{lcm}(c_f, c_g)}{c_f}, \tag{14.35}$$

$$k_{c_g} := \frac{\operatorname{lcm}(c_f, c_g)}{c_g}. \tag{14.36}$$

Then, $f_p^{\vee} \overline{\otimes} g_p^{\vee}$ is again a function of $\mathcal{U}$ with

$$T_{\overline{\otimes}_{pp}} = \inf\left\{ t \geq T_f + T_g \mid f_p^{\vee} \overline{\otimes} g_p^{\vee}(t) \geq f(T_f) + g(T_g) + \operatorname{lcm}(c_f, c_g) \right\}, \tag{14.37}$$

$$d_{\overline{\otimes}_{pp}} = \min\left(\frac{d_f}{c_f}, \frac{d_g}{c_g}\right) \cdot \operatorname{lcm}(c_f, c_g) = \min\left(k_{c_g} \cdot d_g, k_{c_f} \cdot d_f\right), \tag{14.38}$$

$$c_{\overline{\otimes}_{pp}} = \operatorname{lcm}(c_f, c_g). \tag{14.39}$$

A proof is provided in Appendix F.4. The theorem provides alternative UPP properties for $f_p^{\vee} \overline{\otimes} g_p^{\vee}$ through the isomorphism that links this computation to $f_{\downarrow,p}^{-1} \otimes g_{\downarrow,p}^{-1}$. For $d_{\overline{\otimes}_{pp}}$ and $c_{\overline{\otimes}_{pp}}$, we can therefore combine these results together with the *direct* approach to leverage from both. Let $d'_{\overline{\otimes}_{pp}}$ be the result from Equation (3.18), and $d''_{\overline{\otimes}_{pp}}$ be the result of Equation (14.38). Then we can use

$$d_{\overline{\otimes}_{pp}} = \min\left(d'_{\overline{\otimes}_{pp}}, d''_{\overline{\otimes}_{pp}}\right), \tag{14.40}$$

$$c_{\overline{\otimes}_{pp}} = d_{\overline{\otimes}_{pp}} \cdot \max\left(\frac{c_f}{d_f}, \frac{c_g}{d_g}\right). \tag{14.41}$$

For $T_{\overline{\otimes}_{pp}}$, however, we have that the alternative expression (Equation (14.37)) cannot be computed *a priori*, i.e., without performing the computation of $f_p^\vee \overline{\otimes} g_p^\vee$ first. However, we can still leverage its insight to optimize the convolution as we compute it, since we can check the result and cut away any part of the *by-sequence* convolution with value over $T_{\otimes_p^{-1}} + d_{\otimes_p^{-1}}$. This is shown, together with the rest of the optimization technique, later on in Section 14.4.1.

The isomorphism can be further explored as in the following results, to discuss the *cuts* of f and g required for the computation.

Corollary 14.33. *Let f and $g \in \mathcal{U}$ which are neither UC nor UI, and are right-continuous and non-decreasing in $\left[T_f, +\infty\right[$ and $\left[T_g, +\infty\right[$, respectively. Moreover, let f and g satisfy the assumptions of Lemma 14.11, such that $T_{f_{\downarrow,p}^{-1}} = f(T_f)$, $T_{g_{\downarrow,p}^{-1}} = g(T_g)$.*

Let $k_{c_f} := \frac{\mathrm{lcm}\left(c_f, c_g\right)}{c_f}$, $k_{c_g} := \frac{\mathrm{lcm}\left(c_f, c_g\right)}{c_g}$. Then, to compute $f_p^\vee \overline{\otimes} g_p^\vee$ via the (min, +) *isomorphism* (*Equation* (14.26)), *it is sufficient to use $S_{f_p^\vee}^{I'_{f_p^\vee}}$ and $S_{g_p^\vee}^{I'_{g_p^\vee}}$ with*

$$\begin{aligned} I_{f_p^\vee} &= \left[T_f, T_f + 2 \cdot k_{c_f} \cdot d_f\right], \\ I_{g_p^\vee} &= \left[T_g, T_g + 2 \cdot k_{c_g} \cdot d_g\right]. \end{aligned} \tag{14.42}$$

A proof is provided in Appendix F.4. Corollary 14.33 states that, instead of computing $f_p^\vee \overline{\otimes} g_p^\vee$ using the intervals $I_{f_p^\vee}$ and $I_{g_p^\vee}$ described in Proposition 3.32, by exploiting the isomorphism we can use instead the alternative expressions provided by Equation (14.42). As before, these intervals can be much smaller, and therefore much more efficient to do computations with, since their size depends on $\mathrm{lcm}\left(c_f, c_g\right)$ rather than $\mathrm{lcm}\left(d_f, d_g\right)$. However, this link is proved by using the (min,+) convolution, thus the proof does not establish that we can use these intervals as a direct replacement within the (max,+) convolution. Moreover, replacing *both* intervals is something we could already do (through the *inverse* approach) while in Section 14.1 we have shown, through an example, that it may be the case where, e.g., $f_p^\vee$ has a smaller interval using the *direct* approach and $g_p^\vee$ through the *inverse* approach – we would thus prefer to be able to *mix and match* these intervals to minimize the cuts taken of both functions. We address these issues in the following theorem.

Theorem 14.34 (Mix and Match ((max,+) Convolution)). *Let f and $g \in \mathcal{U}$ which are neither UC nor UI, and are right-continuous and non-decreasing in $\left[T_f, +\infty\right[$ and $\left[T_g, +\infty\right[$, respectively. Moreover, let f and g satisfy the assumptions of Lemma 14.11, such that $T_{f_{\downarrow,p}^{-1}} = f(T_f)$, $T_{g_{\downarrow,p}^{-1}} = g(T_g)$. Let $I_{f_p^\vee}, I_{g_p^\vee}$ be the intervals sufficient to compute $f_p^\vee \overline{\otimes} g_p^\vee$ according to Proposition 3.32, and let $I'_{f_p^\vee}, I'_{g_p^\vee}$ be the intervals sufficient to compute $f_{\downarrow,p}^{-1} \otimes g_{\downarrow,p}^{-1}$ according to Corollary 14.33.*

Then, $I_{f_p^\vee} \cap I'_{f_p^\vee}, I_{g_p^\vee} \cap I'_{g_p^\vee}$ are sufficient intervals to compute $f_p^\vee \overline{\otimes} g_p^\vee$.

A proof is provided in Appendix F.4. The above theorem states that, given different cut intervals for $f_p^\vee$ and $g_p^\vee$ computed with the two methods, we can pick the shortest of each pair for our computations. We recall that the (max,+) *by-sequence* convolution complexity is – similarly to the (min,+) one (Section 3.4.2)

$$\mathcal{O}\left(n\left(S_{f_p^\vee}\right) \cdot n\left(S_{g_p^\vee}\right) \cdot \log\left(n\left(S_{f_p^\vee}\right) \cdot n\left(S_{g_p^\vee}\right)\right)\right),$$

being able to use $I_{f_p^\vee}, I'_{g_p^\vee}$ allows us then to outperform both methods. This is confirmed empirically in Section 14.4.2, where we show the performance improvements obtained.

14.4.1 Improved algorithm for (max,+) convolution

We can apply the previous results to improve the (max,+) convolution algorithm, similarly to what is shown in Section 14.3.1. Though, one notable difference is that the results discussed in Section 14.4 apply to functions which satisfy the assumptions of Lemma 14.11. However, as pointed out in Remark 14.12 and assuming it is $f_p^\vee$ that does not satisfy the Lemma, we can use T_f^{**} in place of T_f to obtain a new decomposition of f that does, i.e., $f_p^\vee = f^\vee_{\left[T_f, T_f^{**}\right[} \vee f^\vee_{\left[T_f^{**}, +\infty\right[}$, where $f^\vee_{\left[T_f^{**}, +\infty\right[}$ satisfies Lemma 14.11. We show this process in Algorithm 12.

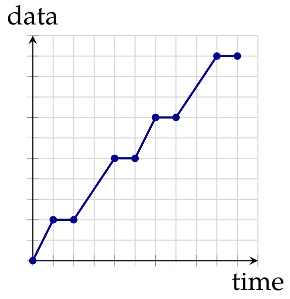
As in the (min,+) case, Algorithm 12 emphasizes that we can improve the cuts of both the operands and the result using the improved parameters. However, implementing the same filtering optimization on the *by-sequence* (max,+) convolution, skipping unnecessary *by-element* convolutions both "horizontally" and "vertically", can be tricky. In fact, since the (max,+) convolution computes an *upper* envelope, applying the vertical filter would remove the elements that belong to the result, leaving incorrect elements in.

Finally, the heuristic algorithm, that chooses between *direct* and *inverse* by-sequence convolution, applies mostly unchanged for the (max,+) convolution as well.

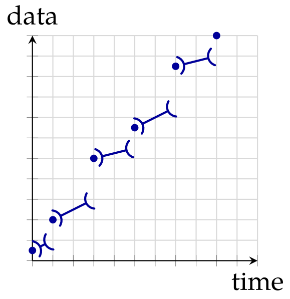
14.4.2 Performance evaluation

In this section, we discuss the performance improvements introduced by the *isospeed* approach for (max,+) convolution. Most of the observations from Section 14.3.2 also apply in this case – the main difference being only that curves are now right-continuous rather than left-continuous. Hence, the three shapes of curves we consider in our experiments are those shown in Figure 14.12. We run the experiments on System 2, using randomly generated parameters for the shapes discussed above.

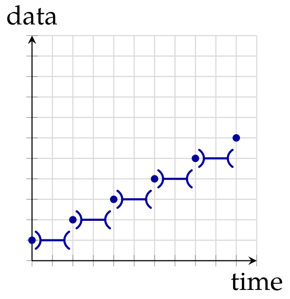
The results of our tests are shown in Figures 14.13 to 14.15. We note that the same considerations apply, mostly unchanged, in this case too. We can observe, in fact, that the *isospeed* approach improves the performance over both methods, aside



(a) *Horizontal* curve.

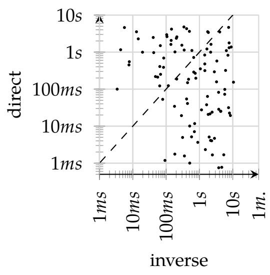


(b) *Vertical* curve.

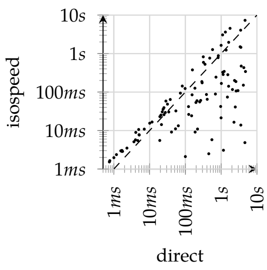


(c) *Balanced* curve.

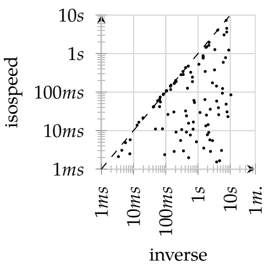
Figure 14.12: Shapes used for performance evaluation of (max,+) convolution.



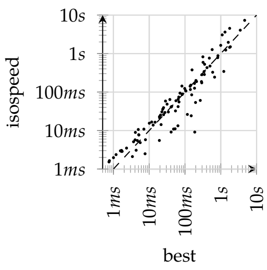
(a) *inverse* vs. *direct*.



(b) *direct* vs. *isospeed*.

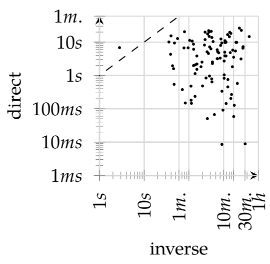


(c) *inverse* vs. *isospeed*.

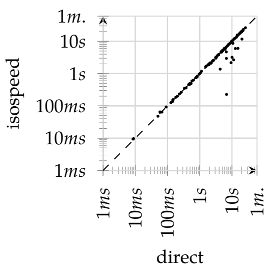


(d) *isospeed* vs. *best*.

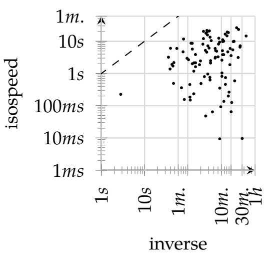
Figure 14.13: Performance comparison of the three algorithms for the (max,+) convolution of *balanced* curves.



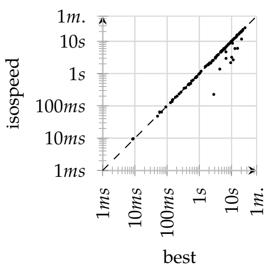
(a) *inverse* vs. *direct*.



(b) *direct* vs. *isospeed*.



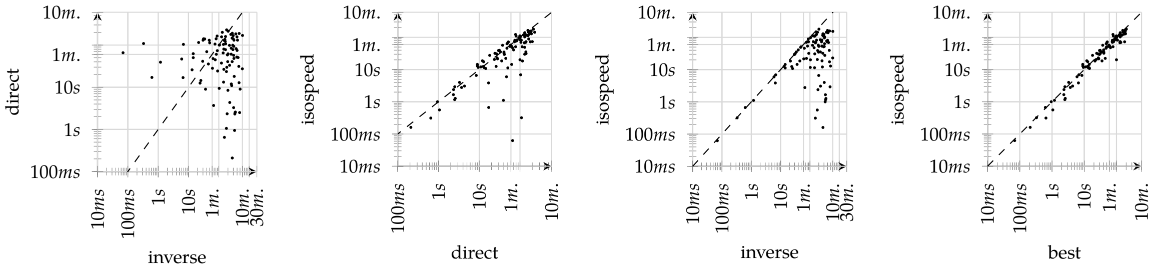
(c) *inverse* vs. *isospeed*.



(d) *isospeed* vs. *best*.

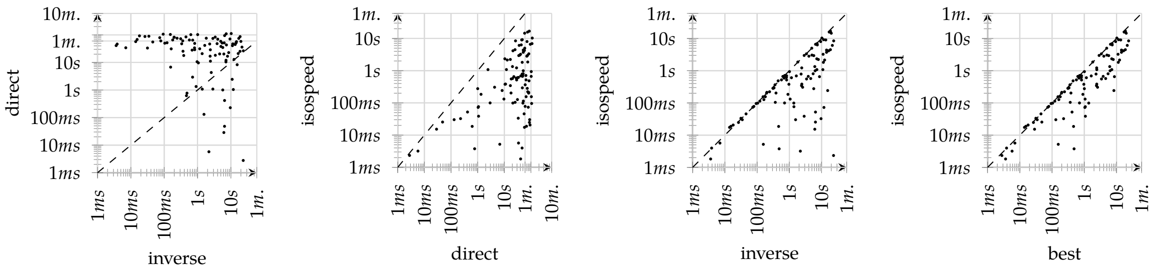
Figure 14.14: Performance comparison of the three algorithms for the (max,+) convolution of *vertical* curves.

from an overhead effect (similar to the one discussed in Section 14.3.2) that appears negligible compared to the benefits obtained in most cases. Moreover, we notice that the improvement happens over both the *direct* and *inverse* approaches, even in cases when one was already better than the other. Thanks to Theorem 14.34, in fact, we select the best cut for both operands independently.



(a) *inverse* vs. *direct*. (b) *direct* vs. *isospeed*. (c) *inverse* vs. *isospeed*. (d) *isospeed* vs. *best*.

Figure 14.15: Performance comparison of the three algorithms for the (max,+) convolution of *horizontal* curves.



(a) *inverse* vs. *direct*. (b) *direct* vs. *isospeed*. (c) *inverse* vs. *isospeed*. (d) *isospeed* vs. *best*.

Figure 14.16: Performance comparison of the three algorithms for (max,+) convolution. Operands are *horizontal* curves, and their parameters are set so that comparing extension multipliers is inconclusive.

Algorithm 12 Improved pseudocode for (max,+) convolution of periodic parts

Input Functions $f_p^{\vee}$ and $g_p^{\vee}$, which are both right-continuous and non-decreasing over their respective support.

Return Their (max,+) convolution $f_p^{\vee} \overline{\otimes} g_p^{\vee}$.

```
1:  if $f_p^{\vee}, g_p^{\vee}$ are not right-continuous and non-decreasing then
2:      Continue with the usual algorithm (see Algorithm 8)
3:  else
4:      if $f_p^{\vee}$ or $g_p^{\vee}$ do not satisfy Lemma 14.11 then
                                        ▷ We assume both do not, for simplicity
5:          Decompose again $f_p^{\vee} = f^{\vee}_{\left[T_f, T_f^{**}\right[} \vee f^{\vee}_{\left[T_f^{**}, +\infty\right[}$
6:          Decompose again $g_p^{\vee} = g^{\vee}_{[T_g, T_g^{**}[} \vee g^{\vee}_{[T_g^{**}, +\infty[}$
7:          Compute $\overline{\otimes}'_{tt}$, $\overline{\otimes}'_{tp}$ and $\overline{\otimes}'_{pt}$ as usual
8:          Compute $\overline{\otimes}'_{pp} = f^{\vee}_{\left[T_f^{**}, +\infty\right[} \overline{\otimes} g^{\vee}_{[T_g^{**}, +\infty[}$ using the improved algorithm
    below
9:          Return $\overline{\otimes}'_{tt} \vee \overline{\otimes}'_{tp} \vee \overline{\otimes}'_{pt} \vee \overline{\otimes}'_{pp}$
10:     else
11:         Compute $k_{c_f} \overset{(14.35)}{=} \frac{\mathrm{lcm}\left(c_f, c_g\right)}{c_f}$ and $k_{c_g} \overset{(14.36)}{=} \frac{\mathrm{lcm}\left(c_f, c_g\right)}{c_g}$
12:         Compute $d = \max\left(\mathrm{lcm}(d_f, d_g), \max\left(k_{c_f} d_f, k_{c_g} d_g\right)\right)$
13:         Compute $c = d \cdot \max\left(\frac{c_f}{d_f}, \frac{c_g}{d_g}\right)$
14:         Compute $T_1 = T_f + T_g + d$
                $I_{f_p} = \left[T_f, T_f + \max\left(\mathrm{lcm}(d_f, d_g), k_{c_f} d_f\right)\right[,$
                $I_{g_p} = \left[T_g, T_g + \max\left(\mathrm{lcm}(d_f, d_g), k_{c_g} d_g\right)\right[$
15:         Compute $S_{\overline{\otimes}_{pp}}^{I_{\overline{\otimes}_{pp}}} = S_{f_p}^{I_{f_p}} \overline{\otimes} S_{g_p}^{I_{g_p}}$
16:         Compute $T_2 = \sup\left\{t \geq T_f + T_g \mid S_{\overline{\otimes}_{pp}}^{I_{\overline{\otimes}_{pp}}}(t) \leq f(T_f) + g(T_g) + \mathrm{lcm}(c_f, c_g)\right\}$
17:         Compute $T = \max(T_1, T_2)$
18:         Remove from $S_{\overline{\otimes}_{pp}}^{\overline{\otimes}_{pp}}$ elements whose support is after $T + d$
19:         $h_{pp} := f_p \overline{\otimes} g_p = \left(S_{\overline{\otimes}_{pp}}^{\overline{\otimes}_{pp}}, T, d, c\right)$
20:     end if
21: end if
```

Chapter 15

Comparison With RTC Toolbox

In this chapter, we compare *Nancy* with *RTC Toolbox*, in particular on the functional features and design choices related to the numerical types used. We discuss, in particular, how the inaccuracies of floating point numbers affect the runtime, causing it to grow unpredictably, as well as the ability to assess properties such as left- or right-continuity.

The *RTC Toolbox* [WTa; Wan06] is a publicly available Java-based library, also available as a MATLAB toolbox, that implements many (min,+) and (max,+) algebra operators. It is designed, as the name suggests, for Real-Time Calculus studies, and provides also functions for Modular Performance Analysis (MPA). Through the MATLAB environment, one can use it to build curves, compute expressions, and plot their results – *Nancy* provides a similar experience with .NET Interactive Netbooks. Work [SB17] integrated *RTC Toolbox* with *NCorg DNC*, leveraging the shared Java runtime. However, the source code of *RTC Toolbox* is not publicly available, which impedes researchers from extending its functionalities, verifying its correctness or fixing its bugs.

For example, [ZHLC20] notes that the toolbox does not provide a function for computing the time of intersection of an arrival and a service curve, which complicates the implementation of studies relying on it. Moreover, the toolbox provides only one inversion operator, with the `rtcinvert` function. The documentation does not clarify whether it conforms to lower or upper pseudoinverse. Work [PLSK11], which first observed the runtime improvements involved with the isomorphism between (min,+) and (max,+), notes that many steps needed to be implemented *outside* the toolbox and, combined with lack of insights on its internal implementation, the authors were therefore unable to discern the cause of the observed improvements. Moreover, the *RTC Toolbox* requires new curves to be constructed with an integer period length, most likely in order to address some issues that we discuss below. On one hand, we believe this to be a very restrictive assumption, which leads to additional work in order to study a system which does not fit this model (e.g., a heterogeneous system with multiple clock domains). On the other hand, as we show, this is not enough to preserve numerical stability.

As for its mathematical model, *RTC Toolbox* works on Variability Characterization Curves (VCC), which appear to be very similar to UPP ones. However, [BT08] notes that the two classes treat discontinuities differently, as it seems from the documentation that VCC are *assumed* to be left- or right-continuous based on the operation being used.[1] Moreover, [RRB21] remarks that the UPP and VCC models were never formally compared.

Finally, we remark that *RTC Toolbox* uses `double`s as its base numeric type. We believe this to be a reason of concern, and the main cause of some unexpected behaviors that we observed – which include non-terminating computations. Note though that, due to the aforementioned lack of source code, we can validate neither the toolbox nor these statements, which should therefore be taken as conjectures.

In the rest of this chapter, we discuss this point further, and provide a comparison, albeit superficial, with *Nancy* – highlighting the computational instability risks in *RTC Toolbox* which are, to the best of our understanding, to be ascribed to its mathematical model and numerical type.

15.1 Floating point numbers

First, we recall what a *floating point* number is – and what is not. Standard IEEE 754 [IEEE754] defines both their numerical representation and algorithms, which can be, generally speaking, ascribed to the following design goals: implement *fast*, *approximate* operations, to be supported directly in hardware.

Simplifying for the sake of discussion, they can be seen as numbers in the form $\pm b \cdot 2^m$, where the base $b \in \mathbb{N}$ and the mantissa $m \in \mathbb{Z}$ have pre-determined bit lengths. Note that such a number is obviously in $\mathbb{Q}$, since numbers in $\mathbb{R} \setminus \mathbb{Q}$ do not have a finite number of digits in *any* base.

As mentioned, the standard does not aim for accurate operations, rather for acceptable error margins for the operations directly supported by the standard. To explain this point more formally, let $x \in \mathbb{R}$. Then its representation as floating point number is $x_\varepsilon \in \mathbb{Q}$, such that $x - \varepsilon \leq x_\varepsilon \leq x + \varepsilon$. Moreover, computing the sum, multiplication, division, etc. the IEEE 754 standard specifies the error ε introduced by those. On one hand, keeping track of these errors, their propagation and the effect on the overall result, so that it can be still trusted within a desired tolerance, is a typical, well-understood problem in engineering, physics and so on – after all, the very concept of error is unavoidable when one has to deal with measurements. On the other hand, if one does not carefully consider this, numerical inconsistencies

[1] "However, the various functions of the RTC Toolbox will interpret a discontinuity correctly as either left- or right-continuous depending on the context of the curve." [WTb]

can occur. For example, it is easy to create, in many languages, instances of computations involving integer numbers and results that can, in a few iterations using floating point operations, yield a non-integer number instead, e.g., incrementing an integer with `1.0` to eventually obtain a non-integer.

The other objective of the standard, efficiency, is very important as well: it is easy to verify that the same program using floating point computations, instead of custom types like our `LongRational` and `BigRational` ones, will reap the benefits of mature hardware support and run much faster. To give a perspective of this difference, we created a simple benchmark (Listing 15.1) which compares the performance of these numerical types for the computation of the intersection between two segments. The

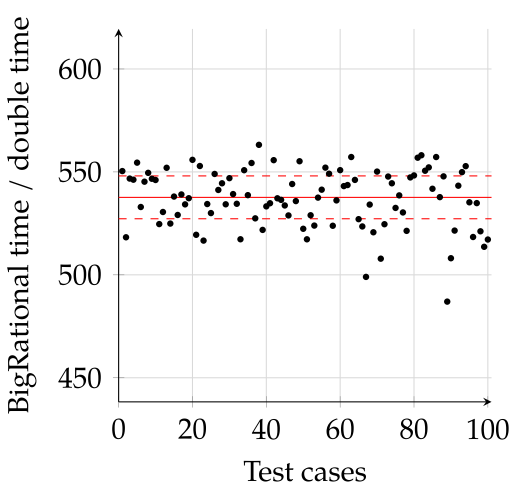
Listing 15.1 Simple benchmark for a perspective on performance difference between numerical types.

```
// T is either float, BigRational or LongRational
public T Benchmark<T>(int x_0, int y_0, int r, int y_1)
{
    T x_0_T = (T) x_0;
    T y_0_T = (T) y_0;
    T r_T = (T) r;
    T y_1_T = (T) y_1;

    var intersection = x_0_T + ((y_1_T - y_0_T) / r_T);
    return intersection;
}
```

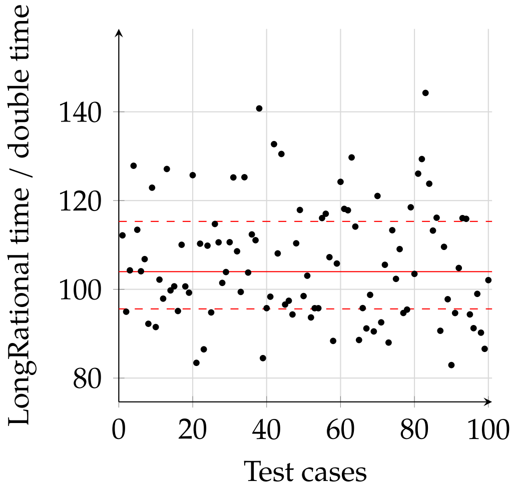
results of the comparison are in Figure 15.1, which highlights the ratio of computation time between our custom rational types and `double` (the red lines mark the median, first and third quartile). These results show a difference of up to 550x between using `double` and `BigRational`. While this benchmark is focused only the individual operation and executed entirely in C#, one can expect the same order of magnitude of difference when the same computation is done in *RTC Toolbox* – that is, until the approximation issues start interfering.

15.2 Floating point approximations and hyper-period explosion

As mentioned, we believe that the use of floating point in the implementation of *RTC Toolbox* is in fact the root cause for the issues that we observed and discuss below. VCC curves have, like UPP ones, a period length, and operations such as the (min,+) convolution will, at some point, compute the hyper-period of its operands, $\mathrm{lcm}(d_f, d_g)$. However, the least common multiple is not a linear operator, and is



(a) Ratio of computation time with `BigRational` vs. `double`.



(b) Ratio of computation time with `LongRational` vs. `double`.

Figure 15.1: Comparison of computation time for intersections between segments, by numerical type used.

highly susceptible to inaccuracies in its operands: it is not simply bounded by a tolerance, like other operations in the IEEE 754 standard are. One can see it with the simple example of 1000 ± 1: while $\operatorname{lcm}(1000, 1000) = 1000$, 1000 and 1001 are coprime, hence $\operatorname{lcm}(1000, 1001) = 1001000$. While the tolerance of an actual `double` implementation would make *this* example highly unlikely, these kinds of issues can still occur, depending also on how the lcm is implemented.

Technical report [SPLT10, Section 2.1.1] mentions, in fact, something to this effect: it states that "due to cutoff errors when using floating point data types" it is often the case that the resulting curve has a period length *longer* than the theoretical $\operatorname{lcm}(d_f, d_g)$. As mentioned, likely to abate this issue, the *RTC Toolbox* requires new curves to be constructed with an integer period length. We show that this assumption, while highly restrictive, is not sufficient to avoid the issue.

In the following experiments we define curves based on the static window flow control, which we discussed in Section 6.1 and provided optimizations for in Chapter 13, and compute their (min,+) convolution, i.e.,

$$\overline{\beta_{r_1,d_1} + h_1} \otimes \overline{\beta_{r_2,d_2} + h_2}.$$

We compute the above convolution using both *RTC Toolbox* and *Nancy*. Note that to do so, we limit parameters to be integers. A relevant example, for which we provide the code in [ZSa], occurs with the following parameters:

```
int delay_1 = 17*19;
int height_1 = 17*19;
int rate_1 = 4000*17*19;
```

```
    int delay_2 = 20;
    int height_2 = 20;
    int rate_2 = 5000*20;
```

While the *Nancy* implementation terminates in 22 seconds (*without* any subadditive optimization involved, using System 4), the *RTC Toolbox* does not terminate within 24 hours (using System 3) and, from the appearing lack of memory activity, we speculate it would never do so.

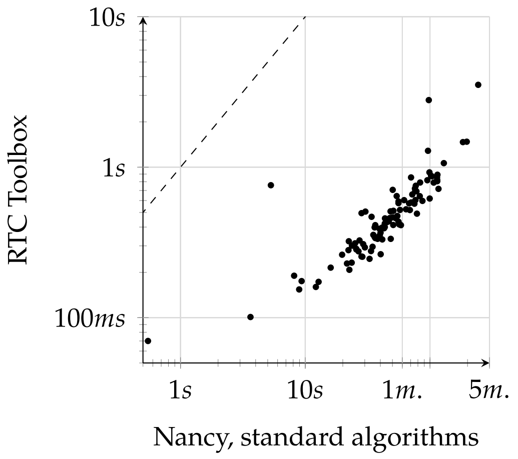
As expected from the comparison of numerical types in the previous section, we observed that in most cases where these anomalies do not arise, *RTC Toolbox* would perform the computation faster than *Nancy* under similar conditions, i.e., single-threaded and without any of the optimizations discussed in Part II. However, it is easy to show that *RTC Toolbox* becomes more unstable the larger the involved numbers are.

Using the same shape of curves as in the previous example, we setup an experiment comprised of a set of 100 convolutions of 5 curves each, with parameters h, d and r randomly generated as integers between 1 and 100. We then run the experiment using both *RTC Toolbox* and *Nancy* (representation minimization always on, with and without subadditive optimizations), with each convolution capped at 10 minutes of runtime, using System 5. If no anomaly was observed, we multiplied all parameters by 10 and rerun the experiment – and so on until said anomalies would occur. We did not have to try for long, as they appeared already at the x1000 step.

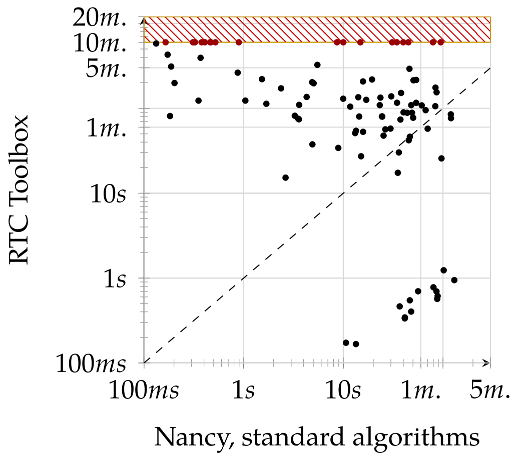
Figures 15.2 and 15.3 show the result for the last two steps of the experiment, i.e., where the randomly generated parameters were multiplied by 100 vs. when the same were multiplied by 1000. In Figure 15.2a we see a positive scenario for *RTC Toolbox*, where it reaps the benefit of the hardware support for `double` operations to achieve 100x faster computation when running – we assume – a similar algorithm to the one run by *Nancy*. However, Figure 15.3a shows that the algorithmic optimizations provided in this thesis have a large effect on these results.

Figure 15.2b paints a very different scenario, though: in most cases, *RTC Toolbox* sees its computation grow much more than those of *Nancy*, reversing the situation observed before already without optimizations. In particular, we see cases that take a second or less with *Nancy*, and exceed the 10 minutes timeout for *RTC Toolbox*. The effects of the subadditive optimizations, as Figure 15.3b shows, only further emphasize the benefits of *Nancy* in this scenario.

We note that the approximation issue does not affect just the computation of the lcm. Consider a VCC function f, then for some threshold $\varepsilon > 0$, f_ε is what is actually stored by *RTC Toolbox*, which is such that given $t \in \mathbb{R}_+$, it computes $f_\varepsilon(t_\varepsilon)$, where $t - \varepsilon \leq t_\varepsilon \leq t + \varepsilon$ and $f(t_\varepsilon) - \varepsilon \leq f_\varepsilon(t_\varepsilon) \leq f(t_\varepsilon) + \varepsilon$. In other words, we highlight that both the time of sampling and the value taken by the function, being both represented with floating point numbers, are subject to approximation errors.

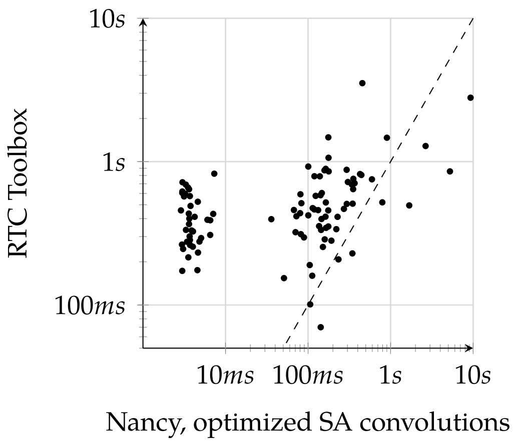


(a) Parameters x100

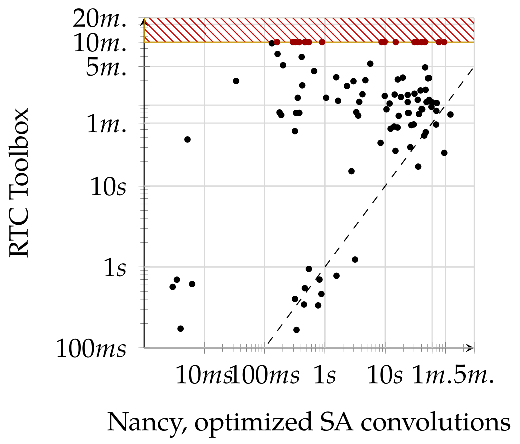


(b) Parameters x1000

Figure 15.2: Comparison of Nancy and RTC Toolbox for chained convolutions.



(a) Parameters x100



(b) Parameters x1000

Figure 15.3: Comparison of Nancy and RTC Toolbox for chained subadditive convolutions.

On one hand, to the best of our knowledge, it has never been formally discussed how such approximation affects (min,+) and (max,+) operations, e.g., with which ε one may bound the result of $f_\varepsilon \otimes g_\varepsilon$. On the other hand, these effects can be observed in practice too. In fact, we observed that the convolutions from the experiment above often return curves such that, given a non-constant segment ending at t_e, followed by a constant segment starting at t_e, $\lim_{t \nearrow t_e} f(t) > f(t_e)$. Thus, as shown visually exaggerated in Figure 15.4, the resulting curve is neither left-continuous nor non-decreasing, even though the inputs are.

These issues would also impede the implementation of some features of *Nancy*, such as checks for left- and right-continuity based on the values of the representation

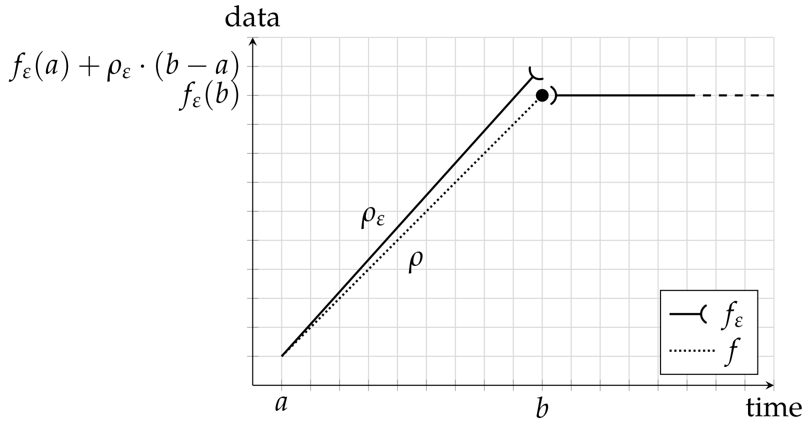


Figure 15.4: Visually exaggerated example of approximation affecting curve properties. The actual function f, shown dotted, is non-decreasing and continuous, but since $f_\varepsilon(a) + \rho_\varepsilon \cdot (b - a) > f_\varepsilon(b)$, it appears not to.

at hand – rather than by assumption.

15.3 Conclusion

We highlighted two particular differences between *RTC Toolbox* and *Nancy*. On one hand, there are functional differences such as richness of API, extensibility, open-source and verifiability; on the other hand, the two libraries have different numerical models which affect their performance and accuracy.

However, a thorough comparison will have to also consider the maturity of the two libraries and their use. In fact, *RTC Toolbox* has seen wide use in the research community, with tools such as *NCorg DNC* [BS14] and *CyNC* [SSH07] built using it, while *Nancy*, being a quite recent endeavor, has only seen applications in our own work, e.g., [ACSZ20; RSLSSZZAH21; AFGSZ22].

As shown, a synthetic benchmark, such as convolution of randomly generated functions, will appear skewed based on the numerical properties of the generated cases, i.e., it may be skewed in favor of *RTC Toolbox* if the numbers are "simple" enough, in favor of *Nancy* if not. Thus, an interesting question is how are these tools used, and how frequently such differences may arise in practice.

A fair comparison should therefore focus on a library of "real world" test scenarios, i.e., examples of studies (and numbers) that actually occur in the practical cases these tools may be used in.

Chapter 16

Conclusion and Future Works

In this thesis, we presented our work on the algorithmic aspects of Deterministic Network Calculus. We showed, through use case examples, that mature software support for (min,+) and (max,+) algebra is needed for the advancement of DNC research, and that the set of functions $\mathcal{U}$ is a good, general model to be implemented is such software – thanks, also, to its use of rationals.

We then discussed the requirements and design goals of such software, which we then developed into *Nancy*, an open source, extensible and efficient library. We discussed how such properties are achieved through its software architecture and the techniques used – including, e.g., generators and immutable objects – and the variety of use cases supported by its rich API.

Using then such properties for our own research, we introduced novel formal results, that we proved formally and implemented in *Nancy*, from the extension of UPP toolbox to include pseudoinverses and composition, to the introduction of optimization algorithms such as representation minimization, improved (min,+) convolution for subadditive curves, improved (min,+) and (max,+) convolution using their isomorphism. Lastly, we compared the functional features and design choices of *Nancy* and *RTC Toolbox*, discussing the issues with the use of floating point numerical types and highlighting instances in which this caused anomalies in the *RTC Toolbox* runtime.

The result of this contribution is a software library, *Nancy*, and a collection of new algebraic results implemented in said library. Due to the user-centered design goals we set for this project, rather than *just* demonstrating the results of our research on the algorithmic properties of DNC, we believe *Nancy* will be a useful tool for future research in the space of deterministic worst-case analysis, on both networks and real-time systems. It implements a mathematical model that can be broadly adapted to many use cases, some of which we used as examples in this thesis, and it does so in an organized manner, providing extensive documentation and verifiability, so that it can be easily included in existing and future research projects with minimal attrition. The algorithmic improvements, provided to the user by default, ensure that the library achieves state-of-the-art performance, improving over baseline algorithms by several orders of magnitude, with a strong formal background validating its results. *Nancy* also provides the user with many knobs for fine-tuning, to optimize the many tradeoffs such an implementation incurs into. Our improved algebraic and algorithmic results allow convolution - which is perhaps the most frequent DNC operation - to run considerably faster, by orders of magnitude in several cases. Improving a frequent operation this much is not just a matter of increased efficiency: especially in conjunction with a usable software tool, it allows scientists and practitioners working on DNC to perform studies that were considered beyond the realm of tractable, so far. An example is the case of flow-controlled networks (Section 13.2.3), where our results allow one to study non-trivial tandems and obtain their end-to-end equivalent service curve. The improvements brought about by the isospeed algorithms (Chapter 14) are of similar impact. It is our hope that these results will stimulate new research in the future.

Of course, such a project is far from being "finished". While we focused mainly on results involving convolutions – likely the most used operation in DNC studies – there may be room to improve the performance of other operations, by either discovering inefficiencies in the current algorithms or providing completely novel results. For example, in the provided algorithms for pseudoinverses and composition, we did not find opportunities for obvious parallelization, while the non-obvious ones would require time for proper comparison and evaluation of tradeoffs. Another operator for which we found challenging use cases is the horizontal deviation, calling for further investigation and profiling, seeking algorithmic or implementation improvements, or both. The algorithmic improvements we provided for the (min,+) convolution of subadditive curves may, as suggested by their general symmetrical behavior, likely to apply also to (max,+) convolution of superadditive curves. On the same note, [Lie17] provides an isomorphism property between the (min,+) and (max,+) deconvolution as well, suggesting that a similar algorithmic improvement could be gained for this operator, too.

As for the library itself, many extensions appear feasible and further beneficial to the research community. Works such as [SB17] highlighted the benefits of integrating network analysis tools with computational libraries, where both benefit from the focus of scope of the other project. Integrating *Nancy* into existing tools incurs into challenges such as difference of language and runtime, as well as internal mathematical model.

While discussing the optimization for (min,+) convolution of subadditive curves, we highlighted how practical use cases benefit greatly from it since the convolution of two subadditive curves is again subadditive – a property which is simple to reflect in code through our class structure. However, many similar properties in literature apply to larger expressions, which can hardly be fully captured by a single object or operation. An interesting further development is then the extension to *symbolic expressions*, which would allow the user to construct entire expressions ahead of time so that its algebraic properties can be observed "top to bottom" before running computations "bottom to top". This would also provide support for parametric expressions, which are frequently used for numerical solutions, e.g., in [BLMS12; SSB22].

Part III

Appendices

Appendix A

Generator Pattern and *yield*

In this Appendix we present the *generator pattern*, which is an implementation optimization that allows one to minimize the active memory utilization of algorithms iterating over large collections, i.e., the amount of memory that is occupied concurrently during its execution. Moreover, we discuss the language facilities in C#, such as `IEnumerable`, `yield`, *extension methods* and LINQ, to support ease of implementation of this pattern. We first discuss, via examples, the pattern and the issues it addresses. Then, we introduce the facilities in C# that improve its usage.

A.1 The generator pattern via examples

Let `List<int> Range(int start, int n)` be a function that returns a list of n integers starting from `start`; let a `Predicate<int>` be a check that, provided an `int`, returns either `true` or `false`; and let `List<int> Where(List<int> numbers, Predicate<int> p)` be a function that *filters* the integers, returning only those that pass the check `p`. For example, these could be used as in Listing A.1. The code in this example allocates three lists, the first with a thousand elements, the second with half of that, and so on. When we compute the second, the first list fully resides in memory; when we compute the third both the first and the second do; and by the time we get to print the numbers, all elements of all three lists reside in memory. We can see the print is just a linear scan, which processes one number at a time. While it is processing a number, it does not need either its precedent or its successor to be in memory: in fact, a number is only needed when it is its turn to be printed. In other words, the memory utilization of this code is wasteful, and can limit the scalability of the program given more initial elements or filtering steps. We can in fact solve the same problem, storing just one number at a time, if we rewrite the above, e.g., as in Listing A.2. While effective, optimizing the code as done in this example requires one to *understand* what is being done and *alter* the code significantly, implementing something very different with the same result. Moreover, such an approach may be significantly im-

Listing A.1 Example code: filter and then print a large list of numbers.

```
List<int> upToOneThousand = Range(1, 1000);

Predicate<int> isOdd = (n => n % 2 == 1);
List<int> oddUpToOneThousand = Where(upToOneThousand, isOdd);

Predicate<int> isMultipleOfThree = (n => n % 3 == 0);
List<int> isOddAndMultipleOfThreeUpToOneThousand =
    Where(oddUpToOneThousand, isMultipleOfThree);

// print the numbers
foreach(int n in numbers) {
    Console.WriteLine(n);
}
```

Listing A.2 The example rewritten for a more efficient use of memory.

```
Predicate<int> isOdd = (n => n % 2 == 1);
Predicate<int> isMultipleOfThree = (n => n % 3 == 0);
for(int n = 1; n <= 100; n++)
{
    if(isOdd(n) && isMultipleOfThree(n))
        Console.WriteLine(n);
}
```

peded by the software architecture, e.g., if the filters are being applied in multiple steps in different parts of the code; or the use of `Where()` is crucial for the organization of the business logic within the codebase, and disrupting it may require a costly redesign (which is the same, just from the opposite point of view).

The *generator pattern* thus attempts to provide a scheme to avoid the issue above *without* significant changes to how the code is written. The core concept is that instead of a *function*, where a single call returns an entire set, we use an *object* with an internal state and `current()` and `next()` methods which allow iterating over the same elements *allocating one at a time*. In particular, a user would first call `next()` and then access the element using `current()`. For example, we could replace `List<int> Range(int start, int n)` with `RangeGenerator Range(int start, int n)`, implemented as in Listing A.3, which can then be used as in Listing A.4. The pattern shines, though, when we observe that generators can be *composed*. Consider the `Where()` method: it had another list `List<int>` as an argument, which we can now replace with a generator, as shown in Listing A.5. Combining these new functions, we obtain the code in Listing A.6. This is very close to the code we started with in Listing A.1, so much that if we used `var` instead of declaring the types explicitly, they would appear to be identical. Figure A.1 shows how the generators are organized in this example: each

Listing A.3 Example of generator: `Range`.

```
class RangeGenerator : Generator<int> {
    int state;
    int remaining;

    public RangeGenerator(int start, int n) {
        state = start - 1;
        remaining = n;
    }

    public bool next() {
        if(remaining > 0) {
            state++;
            remaining--;
            return true
        }
        else
            return false;
    }

    public int current() {
        return state;
    }
}

Generator<int> Range(int start, int n) {
    return new RangeGenerator(start, n);
}
```

Listing A.4 Use of `Range` generator.

```
    var generator = Range(int start, int n);
    while(generator.next()) {
        Console.WriteLine(generator.current());
    }
```

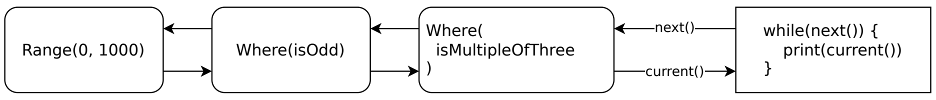


Figure A.1: Schema of generators and print block in Listing A.6

will call `next()` to the generator above, processing its elements one at a time. Thus, we see a reduced memory usage, very close to Listing A.2, the only tradeoff being the generators object themselves.

Listing A.5 Example of generator: Where.

```
class WhereGenerator : Generator<int> {
    Generator<int> other;
    Predicate<int> predicate;

    public WhereGenerator(Generator<int> numbers, Predicate<int> p) {
        other = numbers;
        predicate = p;
    }

    public bool next() {
        bool lastNext;
        do {
            lastNext = other.next()
        } while( lastNext && !predicate(other.current) )

        return lastNext;
    }

    public int current() {
        return other.current();
    }
}

Generator<int> Where(Generator<int> numbers, Predicate<int> p) {
    return new WhereGenerator(numbers, p);
}
```

Listing A.6 The example using the generator pattern: efficient use of memory, while leaving the code organization untouched.

```
Generator<int> upToOneThousand = Range(1, 1000);

Predicate<int> isOdd = (n => n % 2 == 1);
Generator<int> oddUpToOneThousand = Where(upToOneThousand, isOdd);

Predicate<int> isMultipleOfThree = (n => n % 3 == 0);
Generator<int> isOddAndMultipleOfThreeUpToOneThousand =
    Where(oddUpToOneThousand, isMultipleOfThree);

while(isOddAndMultipleOfThreeUpToOneThousand.next()) {
    Console.WriteLine(isOddAndMultipleOfThreeUpToOneThousand.current());
}
```

A.2 The generator pattern in C#

The pattern described above, while not reaching the efficiency of Listing A.2 due to the object overhead, gets really close to it with minor code changes. Moreover, this can be done without changing the code architecture – we note that in many cases the highest obstacle to improving performance does not lie in the availability of better algorithms, but in the cost, in time and effort, to implement them. But as we have shown, there is still an effort to be made to introduce a class architecture that implements the pattern. The C# language provides instead facilities that reduce the above effort, making writing new generators no harder than writing the `List<>` equivalent.

First, we introduce the real type that corresponds to the `Generator<T>` type that we introduced in the example: the `IEnumerable<T>` interface. One would thus define functions such as `IEnumerable<T> Where(IEnumerable<T>, Predicate<T>)` and classes that implement `IEnumerable<T>`.

However, looking closely to Listings A.3 and A.5, we note that most of it is *boilerplate*, i.e., repetitive code that introduces little new information. In fact, the main point of interest of a generator is not how it stores and manages its *state*, which is what the `class` syntax is oriented towards, but how it uses its *parameters* and the *logic* through which it derives the next element. The latter, though, is what a *function* syntax is oriented towards. We can see it with the `Where` filter: we show in Listing A.7 an implementation for its `List<int>` version seen in Listing A.1, which is much simpler than the `Generator<int>` shown in Listing A.5.

Listing A.7 Implementation of `Where()` as seen in Listing A.1.

```
List<int> Where(List<int> numbers, Predicate<int> p) {
    List<int> result = new List<int>();
    foreach(var n in numbers)
        if(p(n))
            result.Add(n);
    return result;
}
```

To be able to write a generator with as simple code as the function in the example above, C# introduces the `yield` keyword. Using the `yield` keyword we can thus focus on the code to return a single element. We use `yield return` to return a value, and `yield break` to signal the exhaustion of the generator. The compiler will then do the rest, translating this code into a `class` with proper state management, where each instance of `yield return` will correspond to `next()` returning `true`, and `yield break` will correspond to `next()` returning false. This not only saves the programmer effort to implement Listing A.5 by themselves, as we show in Listing A.8, but also makes it

Listing A.8 Implementation of `Where()` as a generator, using `yield`.

```
IEnumerable<int> Where(IEnumerable<int> numbers, Predicate<int> p) {
    foreach(var n in numbers)
        if(p(n))
            yield return n;
    yield break; // can be implicit in this case
}
```

Listing A.9 Marking `Where()` as an extension method.

```
public static class WhereExtension {
    public static IEnumerable<T> Where(this IEnumerable<T> elements, Predicate<T> predicate)
    {...}
}
```

Listing A.10 The example of Listing A.6 written using `Where()` as an extension method.

```
var isOddAndMultipleOfThreeUpToOneThousand = Range(1, 1000)
    .Where(n => n % 2 == 1)
    .Where(n => n % 3 == 0);

foreach(var n in isOddAndMultipleOfThreeUpToOneThousand)
    Console.WriteLine(n);
```

feasible to write more complex logic, with branching and nested loops, as a single generator function, which would otherwise be highly error-prone.

A.2.1 Extension methods

One further improvement, particularly useful for this pattern, is the ability to mark user defined methods as *extension methods* of another class, even of a system class, which can be then called as naturally as an instance method. We show this via the example of Listing A.9. Note the `this` keyword before the first argument: it indicates that one can call this method as an instance method of any object of type `IEnum⌋erable<T>`. This allows us to greatly simplify code without actually changing the compiled result. We show this in Listing A.10, which will compile to the same binary as Listing A.6.

It should not surprise the reader, then, to know that LINQ is actually a large set of generator functions, implemented exploiting `IEnumerable<T>` and `yield`, and defined as extension methods for `IEnumerable<T>` itself. This gives the programmer, right out of the box, a large toolset of easy-to-exploit algorithms with reduced allocations, together with the language facilities to efficiently and effectively add their own.

We do so at large in *Nancy*, where multiple algorithms are implemented as generators so that, even though the codebase grows more and more interconnected and a solution such as Listing A.2 unfeasible to implement, the various operators are naturally optimized to minimize the active memory utilization. For example, the `f.Cut(a, b)` method, which returns a `Sequence` containing the values of f in $[a, b[$, is provided also as `f.CutAsEnumerable(a, b)`, which returns the elements of the would-be-sequence one at time, as an `IEnumerable<Element>`. This can be leveraged, for example, in methods such as `Curve.Equivalent()` or `Curve.IsNonDecreasing`, which perform checks iterating one element at a time and may terminate early – using the generator pattern, when that happens we avoid that unnecessary elements are ever computed, allocated or processed.

Appendix B

UPP properties for minimum and (min,+) convolution

We report below the proofs for the properties of minimum and (min,+) convolution for functions in $\mathcal{U}$. These proofs are adapted from those in [BT07; BT08], with a few clarifications of our own.

B.1 Minimum

Proposition 3.10 (Minimum of Functions with the Same Slope). *Let f, g be functions of $\mathcal{U}$, neither of which is UI. If $\rho_f = \rho_g := \rho$, $f \wedge g$ is again a function of $\mathcal{U}$ with*

$$\begin{aligned} T &= \max\left(T_f, T_g\right), \\ d &= \operatorname{lcm}\left(d_f, d_g\right), \\ c &= \rho \cdot d. \end{aligned}$$

Proof. First, we observe that $f \wedge g$ is again a piecewise affine function in $\mathbb{Q}_+ \to \mathbb{Q} \cup \{+\infty, -\infty\}$, since any two functions in $\mathcal{U}$ intersect at a rational point ([BT08, Proposition 6]). Then, for any $t \geq \max\left(T_f, T_g\right)$,

$$\begin{aligned} (f \wedge g)(t + d) &= f(t + d) \wedge g(t + d) \\ &= \left(f(t) + \frac{d}{d_f} \cdot c_f\right) \wedge \left(g(t) + \frac{d}{d_g} \cdot c_g\right) \\ &= (f(t) + \rho \cdot d) \wedge (g(t) + \rho \cdot d) \\ &= f(t) \wedge g(t) + \rho \cdot d. \end{aligned}$$

We observe that, in order to be able to leverage the pseudo-periodicity property for both f and g in this proof:

- it is enough that t is the largest between T_f and T_g, i.e., $t = \max\left(T_f, T_g\right)$;
- it is necessary that $d = \mathrm{lcm}\left(d_f, d_g\right)$, as both $\frac{d}{d_f}$ and $\frac{d}{d_g}$ need to be $\in \mathbb{N}$.

□

Proposition 3.11 (Minimum of Functions with Different Slopes)**.** *Let f, g be functions of $\mathcal{U}$, neither of which is UI. Let, without loss of generality, $\rho_f < \rho_g$, and let $\bar{t} := \frac{M_f - m_g}{\rho_g - \rho_f}$, where $M_f = \sup_{T_f \leq t < T_f + d_f} \left\{f(t) - \rho_f \cdot t\right\}$ and $m_g = \inf_{T_g \leq t < T_g + d_g} \left\{g(t) - \rho_g \cdot t\right\}$. Then, $f \wedge g$ is pseudo-periodic with*

$$\begin{aligned} T &= \max\left(T_f, T_g, \bar{t}\right), \\ d &= d_f, \\ c &= c_f. \end{aligned}$$

Proof. First, we observe that $f \wedge g$ is again a piecewise affine function in $\mathbb{Q}_+ \to \mathbb{Q} \cup \{+\infty, -\infty\}$, since any two functions in $\mathcal{U}$ intersect at a rational point ([BT08, Proposition 6]). Since $\rho_f < \rho_g$, it exists $t^* : \forall t \geq \bar{t}, f(t) \leq g(t)$, hence $f \wedge g(t) = f(t)$ for any $t \geq t^*$. Let $t^* < \max\left(T_f, T_g\right)$, then $f \wedge g$ is UPP from $\max\left(T_f, T_g\right)$. Otherwise, since $t^* \geq \max\left(T_f, T_g\right)$, we can use the UPP properties of f and g to upper-bound t^*. Let $U_f(t)$ be the upper boundary of the pseudo-periodic behavior of f, i.e., the line such that $f(t) \leq U_f(t)\ \forall t \geq T_f$. Let $L_g(t)$ be the lower boundary of the pseudo-periodic behavior of g, i.e., the line such that $g(t) \geq L_g(t)\ \forall t \geq T_g$. Then, t^* is upper-bounded by $\min\left(t \mid U_f(t) \leq L_g(t)\right)$, i.e., $t^* \leq \bar{t} := \frac{M_f - m_g}{\rho_g - \rho_f}$.

Combining both cases, we obtain that $f \wedge g$ is UPP from $T_{f \wedge g} = \max\left(T_f, T_g, \bar{t}\right)$. Moreover, for any $t \geq T_{f \wedge g}$

$$\begin{aligned} (\min(f, g))(t + d_f) &= \min(f(t + d_f), g(t + d_f)) \\ &= f(t + d_f) \\ &= f(t) + c_f \\ &= \min(f(t), g(t)) + c_f, \end{aligned}$$

which concludes the proof. □

B.2 (min,+) convolution

In this section, we provide proofs for the UPP properties of (min,+) convolution of functions in $\mathcal{U}$, starting from the convolutions between *transient* and *periodic* parts. We recall the definition of (min,+) convolution.

Definition 3.12 ((min,+) convolution). Let f and g be functions in $\mathcal{U}$. We define the (min,+) convolution, for all $t \geq 0$, as

$$f \otimes g(t) := \inf_{0 \leq s \leq t} \{f(s) + g(t-s)\} \tag{3.4}$$

Proposition 3.13 ((min,+) Convolution of *Transient* Parts). *Let f and g be functions of $\mathcal{U}$. Introducing the shorthand notation $\otimes_{tt} := f_t^\wedge \otimes g_t^\wedge$, it holds that $\otimes_{tt}$ is again a function of $\mathcal{U}$, and is UI with $\otimes_{tt}(t) = +\infty$ for any $t \geq T_f + T_g$.*

Proof. First, we observe that $\otimes_{tt}$ is again a piecewise affine function in $\mathbb{Q}_+ \to \mathbb{Q} \cup \{+\infty, -\infty\}$, as shown in [BT08, Proposition 7]. Since $f_t^\wedge(t) = +\infty$ for any $t \geq T_f$ and $g_t^\wedge(t) = +\infty$ for any $t \geq T_g$, it follows that for any $t \geq T_f + T_g$ and $0 \leq s \leq t$ we obtain

$$f(s) + g(t-s) = +\infty,$$

hence the inf is in turn $+\infty$, which concludes the proof. □

Proposition 3.14 ((min,+) Convolution of a *Transient* with a *Periodic* Part). *Let f and g be functions of $\mathcal{U}$. Introducing the shorthand notation $\otimes_{tp} := f_t^\wedge \otimes g_p^\wedge$, it holds that $\otimes_{tp}$ is again a function of $\mathcal{U}$ with*

$$\begin{aligned} T_{\otimes_{tp}} &= T_f + T_g, && (3.5)\\ d_{\otimes_{tp}} &= d_g, && (3.6)\\ c_{\otimes_{tp}} &= c_g. && (3.7) \end{aligned}$$

Moreover, $\otimes_{tp}(t) = +\infty$ for any $t < T_g$.

Proof. First, we observe that $\otimes_{tp}$ is again a piecewise affine function in $\mathbb{Q}_+ \to \mathbb{Q} \cup \{+\infty, -\infty\}$, as shown in [BT08, Proposition 7]. Since $f_t^\wedge(t) = +\infty$ for all $t \geq T_f$, we can write, for all $t \geq 0$,

$$(f_t^\wedge \otimes g_p^\wedge)(t) = \inf_{0 \leq s < T_f} (f_t^\wedge(s) + g_p^\wedge(t-s)).$$

Then, for $t \geq T_f + T_g$, if $0 \leq s < T_f$ it follows that $t - s \geq T_g$, and

$$\begin{aligned} (f_t^\wedge \otimes g_p^\wedge)(t + d_p) &= \inf_{0 \leq s < T_f} (f_t^\wedge(s) + g_p^\wedge(t + d_p - s)) \\ &= \inf_{0 \leq s < T_f} (f_t^\wedge(s) + g_p^\wedge(t-s)) + c_g \\ &= (f_t^\wedge \otimes g_p^\wedge)(t) + c_g, \end{aligned}$$

which concludes the proof. □

Proposition 3.15 ((min,+) Convolution of *Periodic* Parts)**.** *Let f and g be functions of $\mathcal{U}$. Introducing the shorthand notation $\otimes_{pp} := f_p^\wedge \otimes g_p^\wedge$, it holds that $\otimes_{pp}$ is again a function of $\mathcal{U}$ with*

$$T_{\otimes_{pp}} = T_f + T_g + \operatorname{lcm}\left(d_f, d_g\right), \tag{3.8}$$

$$d_{\otimes_{pp}} = \operatorname{lcm}\left(d_f, d_g\right), \tag{3.9}$$

$$c_{\otimes_{pp}} = d_{\otimes_{pp}} \cdot \min\left(\frac{c_f}{d_f}, \frac{c_g}{d_g}\right). \tag{3.10}$$

Proof. First, we observe that $\otimes_{pp}$ is again a piecewise affine function in $\mathbb{Q}_+ \to \mathbb{Q} \cup \{+\infty, -\infty\}$, as shown in [BT08, Proposition 7]. It holds for all $t \geq T_f + T_g + d = T_f + T_g + \operatorname{lcm}\left(d_f, d_g\right)$ that

$$\begin{aligned}
& f_p^\wedge \otimes g_p^\wedge (t+d) \\
=& \inf_{0 \leq s \leq t+d} \left\{ f_p^\wedge(s) + g_p^\wedge(t+d-s) \right\} \\
\overset{\left(s \geq T_f, t+d-s \geq T_g\right)}{=}& \inf_{T_f \leq s \leq t+d-T_g} \left\{ f(s) + g(t+d-s) \right\} \\
=& \inf_{T_f \leq s \leq t-T_g} \left\{ f(s) + g(t+d-s) \right\} \\
& \wedge \inf_{t-T_g \leq s \leq t+d-T_g} \left\{ f(s) + g(t+d-s) \right\} \\
\overset{\left(t-T_g \geq T_f+d\right)}{=}& \inf_{T_f \leq s \leq t-T_g} \left\{ f(s) + g(t+d-s) \right\} \\
& \wedge \inf_{T_f+d \leq s \leq t+d-T_g} \left\{ f(s) + g(t+d-s) \right\} \\
\overset{(u=t+d-s)}{=}& \inf_{T_f \leq s \leq t-T_g} \left\{ f(s) + g(t+d-s) \right\} \\
& \wedge \inf_{T_g \leq u \leq t-T_f} \left\{ f(t+d-u) + g(u) \right\} \\
\overset{\left(t-s \geq T_g, t-u \geq T_f\right)}{=}& \inf_{T_f \leq s \leq t-T_g} \left\{ f(s) + g(t-s) + \frac{d}{d_g} c_g \right\} \\
& \wedge \inf_{T_g \leq u \leq t-T_f} \left\{ f(t-u) + \frac{d}{d_f} c_f + g(u) \right\} \\
=& \min\left(f_p^\wedge \otimes g_p^\wedge(t) + \frac{d}{d_g} c_g, f_p^\wedge \otimes g_p^\wedge(t) + \frac{d}{d_f} c_f \right) \\
=& f_p^\wedge \otimes g_p^\wedge(t) + \min\left(\frac{d}{d_g} c_g, \frac{d}{d_f} c_f \right)
\end{aligned}$$

$$=f_p^{\wedge} \otimes g_p^{\wedge}(t) + \min\left(\frac{c_g}{d_g}, \frac{c_f}{d_f}\right) \cdot d.$$

Thus, $f_p^{\wedge} \otimes g_p^{\wedge}$ is Ultimately Pseudo-Periodic from $T_f + T_g + d_{\otimes_{pp}}$ with period length $d_{\otimes_{pp}} = \operatorname{lcm}(d_f, d_g)$ and height $\min\left(\frac{d_{\otimes_{pp}}}{d_g} c_g, \frac{d_{\otimes_{pp}}}{d_f} c_f\right) = \min\left(\frac{c_g}{d_g}, \frac{c_f}{d_f}\right) d$. □

Proposition 3.16 (Sufficient Cuts for (min,+) Convolution of *Periodic* Parts)**.** *Let f and g be functions of $\mathcal{U}$, and let $f_p^{\wedge}$ and $g_p^{\wedge}$ be their respective* periodic *parts. Let the interval $I_{\otimes_{pp}}$ be of the form $\left[T_f + T_g, T_f + T_g + 2 \cdot d_{\otimes_{pp}}\right[$. Then, in order to compute $f_p^{\wedge} \otimes g_p^{\wedge}$ over $I_{\otimes_{pp}}$ it is sufficient to use*

$$\begin{aligned} I_f &= \left[T_f, T_f + 2 \cdot d_{\otimes_{pp}}\right[, \\ I_g &= \left[T_g, T_g + 2 \cdot d_{\otimes_{pp}}\right[. \end{aligned}$$

Proof. The corollary follows directly from the proof of Proposition 3.15. Consider $t = \sup I_{\otimes_{pp}} = T_f + T_g + 2 \cdot d$. Then, we observe that the proof of Proposition 3.15 computes $f(s)$ over $s \in [T_f, T_f + 2 \cdot d]$ and $g(u)$ over $u \in [T_g, T_g + 2 \cdot d]$ □

Proposition 3.17 (Minimum of Decomposed (min,+) Convolution Terms)**.** *Let f and g be functions of $\mathcal{U}$, and let $f = f_t^{\wedge} \wedge f_p^{\wedge}$ and $g = g_t^{\wedge} \wedge g_p^{\wedge}$ be their decomposition in* transient *and* periodic *parts. Let $\otimes_{tt} := f_t^{\wedge} \otimes g_t^{\wedge}$, $\otimes_{tp} := f_t^{\wedge} \otimes g_p^{\wedge}$, $\otimes_{pt} := f_p^{\wedge} \otimes g_t^{\wedge}$, and $\otimes_{pp} := f_p^{\wedge} \otimes g_p^{\wedge}$. Then*

$$f \otimes g = \otimes_{tt} \wedge \otimes_{tp} \wedge \otimes_{pt} \wedge \otimes_{pp}. \tag{3.11}$$

Moreover, $f \otimes g$ is again a function of $\mathcal{U}$ with

$$\begin{aligned} d_{f \otimes g} &= \operatorname{lcm}(d_f, d_g), \\ c_{f \otimes g} &= \min\left(\frac{c_f}{d_f}, \frac{c_g}{d_g}\right) d'_{f \otimes g} = \min\left(\frac{c_f}{d_f}, \frac{c_g}{d_g}\right) \cdot \operatorname{lcm}(d_f, d_g). \end{aligned}$$

Proof. First, we prove the decomposition using the distributivity of the (min,+) convolution.

$$\begin{aligned} f \otimes g &= \left(f_t^{\wedge} \wedge f_p^{\wedge}\right) \otimes \left(g_t^{\wedge} \wedge g_p^{\wedge}\right) \\ &= \left(f_t^{\wedge} \otimes \left(g_t^{\wedge} \wedge g_p^{\wedge}\right)\right) \wedge \left(f_p^{\wedge} \otimes \left(g_t^{\wedge} \wedge g_p^{\wedge}\right)\right) \\ &= \left(f_t^{\wedge} \otimes g_t^{\wedge}\right) \wedge \left(f_t^{\wedge} \otimes g_p^{\wedge}\right) \wedge \left(f_p^{\wedge} \otimes g_t^{\wedge}\right) \wedge \left(f_p^{\wedge} \otimes g_p^{\wedge}\right). \end{aligned}$$

Then, we can derive the UPP properties of $f \otimes g$ using Propositions 3.10 and 3.11 and Propositions 3.13 to 3.15.

Since $\otimes_{tt}$ is UI, it does not affect this discussion. Then, $\otimes_{tp}$ has $\rho_{tp} = \rho_g$, and similarly $\otimes_{pt}$ has $\rho_{pt} = \rho_f$. Lastly, $\otimes_{pp}$ has $\rho_{pp} = \min\left(\rho_f, \rho_g\right)$. Without loss of generality, let $\rho_f > \rho_g$. Then we can derive that $\otimes_{tp} \wedge \otimes_{pp}$ is UPP with $T = \max\left(T_{\otimes_{tp}}, T_{\otimes_{pp}}\right) = T_f + T_g + d_{\otimes_{pp}}$, $d = \operatorname{lcm}\left(d_{\otimes_{tp}}, d_{\otimes_{pp}}\right) = d_{\otimes_{pp}} = \operatorname{lcm}\left(d_f, d_g\right)$, and $\rho = \rho_g$. However, for $\otimes_{pt} \wedge \left(\otimes_{tp} \wedge \otimes_{pp}\right)$ we have that, since $\rho_{\otimes_{pt}} = \rho_f > \rho_g$, to compute its pseudo-period start, we would need to bound the intersection point t^* (Proposition 3.11). Since this depends on the shape of these functions, we cannot predict a valid T *a priori*. We can compute, though, $d_\otimes = d_{\otimes_{tp} \wedge \otimes_{pp}} = \operatorname{lcm}\left(d_f, d_g\right)$ and $c_\otimes = d_\otimes \cdot \min\left(\rho_f, \rho_g\right)$. □

Proposition 3.18 (Convolution of Curves with the Same Slope)**.** *Let f and g be functions of $\mathcal{U}$, with $\rho := \rho_f = \rho_g$. Then $f \otimes g$ is again a function of $\mathcal{U}$ with*

$$\begin{aligned} T &= T_f + T_g + d, \\ d &= \operatorname{lcm}\left(d_f, d_g\right), \\ c &= \rho \cdot d. \end{aligned}$$

The entire convolution can be computed using sequences S_f^I, S_g^I with $I = \left[0, T_f + T_g + 2 \cdot d\right[$, and then computing the convolution of these sequences $S_{f \otimes g}^I$ over the same interval.

Proof. The proof follows the same steps of Proposition 3.17. Since $\rho_f = \rho_g$, we obtain for the last step that $T_\otimes = T_f + T_g + d$. Moreover, it follows that it is sufficient to compute $f \otimes g$ in $I_\otimes = \left[0, T_f + T_g + 2 \cdot d\right[$. Then, since

$$f \otimes g(t) = \inf_{0 \leq s \leq t} \left\{ f(s) + g(t - s) \right\},$$

we derive that $f(s)$ and $g(t - s)$ are computed over $I_f = I_g = I_\otimes$. □

Appendix C

Continuity and $(\cdot,+)$ Convolution

C.1 Note on the non-completeness of $\mathbb{Q}$

Many of the results and proofs from the literature, that here we use and report, use the set of real numbers $\mathbb{R}$ and its *completeness* property, also known as *least-upper-bound* property. This property states that given any set $S \subset \mathbb{R}$, then $\inf S \in \mathbb{R}$, same for $\sup S$. This is in contrast with $\mathbb{Q}$, e.g., $S = \left\{x \in \mathbb{Q} | x^2 \geq 2\right\} \subset \mathbb{Q}$, while $\inf \{S\} = \sqrt{2} \notin \mathbb{Q}$.

Thus, those proof cannot be applied as-is also for functions of $\mathcal{U}$. However, we can prove and use weaker results that are based on the following observation. We defined functions of $\mathcal{U}$ as $\mathbb{Q}_+ \to \mathbb{Q} \cup \{+\infty, -\infty\}$, piecewise and composed of *points* and *segments*, which in turn are defined to have breakpoints, values, limits and slopes in $\mathbb{Q}$ (see Definitions 3.4 and 3.5). It follows that all the infima and suprema operations used, i.e., those in convolutions and pseudoinverses, which use linear combinations of breakpoints and values of such functions, are again in $\mathbb{Q} \cup \{+\infty, -\infty\}$. Moreover, whenever such infima or suprema would be attained for a given t^*, it must be that $t^* \in \mathbb{Q}$.

Lemma C.1. *Let $(t_n)_{n \in \mathbb{N}}$ be a sequence $\in \mathbb{Q}_+$ and let $f \in \mathcal{U}$. Then, for any limit $\in \mathbb{Q}$ of $f(t_n)$ such that $t^* \in \mathbb{R}_+$ attains this limit, i.e., $\lim_{n \to \infty} f(t_n) = f(t^*) \in \mathbb{Q}$, it holds that t^* is already $\in \mathbb{Q}_+$.*

Proof. We proceed with a proof by contradiction. Assume that the rational limit is attained by $t^* \in \mathbb{R}_+ \setminus \mathbb{Q}_+$. As $f \in \mathcal{U}$, there exists $b_i, \rho_i, q_i \in \mathbb{Q}$ such that $f(t^*) = b_i + \rho_i \cdot t^* = q_i \in \mathbb{Q}$. But this is a contradiction, since $\frac{q_i - b_i}{\rho_i} \in \mathbb{Q}$, where we used that a subtraction and a division of rational numbers is, again, rational. □

Thus, we show that the results are mathematically solid also for functions in $\mathcal{U}$.

C.2 Attainability and left-continuity of (min,+) convolution

We can now prove the left-continuity and attainability of (min,+) convolution of left-continuous functions.

Proposition 4.1 (Attainability of (min,+) Convolution). *Let f and g be left-continuous, non-decreasing functions of $\mathcal{U}$. Then, for any $t \in \mathbb{Q}_+$ it exists $s^* \in [0,t]$ such that*

$$f \otimes g(t) = \inf_{0 \leq s \leq t} \{f(s) + g(t-s)\} = f(s^*) + g(t - s^*).$$

In other words, the infimum is always attained in $[0,t]$.

Proof. The proof follows along the lines of [BBL18, p. 48]. Fix $t \geq 0$ and define $F_t(s) := f(s) + g(t-s)$. Let $(s_n)_{n\in\mathbb{N}}$ be a sequence such that $F_t(s_n)$ is decreasing and converges to $\inf_{0\leq s\leq t}\{f(s)+g(t-s)\}$. Note, that the limit $\inf_{0\leq s \leq t}\{f(s)+g(t-s)\}$ is in $\mathbb{Q}$, as it is true for (min,+) convolution of two piecewise affine functions in general [BT08, Proposition 7]. As for all n, it holds that $s_n \in [0,t]$. In other words, we have an infinite sequence on a closed and bounded interval. Therefore, by the Bolzano-Weierstrass theorem for real sequences, there exists a subsequence u_n that converges to $\bar{u} \in [0,t] \subset \mathbb{R}_+$. Moreover, by the Lemma C.1, this limit $\bar{u} \in \mathbb{Q}_+$. By the Monotone Subsequence Theorem, there exists a monotonic subsequence $(v_n)_{n\in\mathbb{N}}$ (of this subsequence u_n) with the same limit $\bar{u}$. Without loss of generality, this sequence is increasing (if this sequence is decreasing, we can exchange the role of f and g and replace v_n with $t - v_n$). As f is left-continuous, $\lim_{n\to\infty} f(v_n) = f(\bar{u})$, and, as g is non-decreasing, $\lim_{n\to\infty} g(t-v_n) \geq g(t-\bar{u})$. Therefore,

$$\lim_{n\to\infty} F_t(v_n) = \lim_{n\to\infty} f(v_n) + g(t-v_n) \geq f(\bar{u}) + g(t-\bar{u}) = F(\bar{u}).$$

Thus, F_t reaches its minimum on $[0,t]$. □

Proposition 4.2 (Left-Continuity of (min,+) Convolution). *Let f and g be left-continuous, non-decreasing functions of $\mathcal{U}$. Then, $f \otimes g$ is left-continuous.*

Proof. The proof follows mostly along the lines of Proposition 3.11 in [BBL18, p. 49].

For all $s \geq 0$, define $F_s : t \to f(s) + g\left([t-s]^+\right)$. The left-continuity of F_s directly follows from that of g and for all $t \geq 0$,

$$\begin{aligned} f \otimes g(t) &= \inf_{0\leq s\leq t} \{F_s(t)\} \\ &= \inf_{s\geq 0} \{F_s(t)\}. \end{aligned}$$

The last equality is because f is non-decreasing. Note that from Proposition 4.1, this is a minimum, and we denote one value by $u_t \in [0,t]$ such that $f \otimes g(t) = F_{u_t}(t)$.

Fix $t \geq 0$ and let t_n be a non-decreasing sequence converging to t. To prove the left-continuity of $f \otimes g(t)$, we need to show that $F_{u_n}(t_n)$ converges to $F_{u_t}(t)$, with $u_n = u_{t_n}$.

By the Monotone Subsequence Theorem, there exists a monotonic subsequence $(v_{t_n})_{n\in\mathbb{N}}$ (of u_n) with the same limit u_t. Without loss of generality, this sequence is increasing (if this sequence is decreasing, we can exchange the role of f and g and replace v_{t_n} with $t_n - v_{t_n}$). Since both sequences, t_n and v_{t_n}, are increasing, we only know that $t_n - v_{t_n}$ converges to $t - u_t$, but not how. The sequence $g\left(t_n - v_{t_n}\right)$ either has an increasing subsequence with $t_n - v_{t_n} \geq 0$ (in this case, $\lim_{n\to\infty} g\left(\left[t_n - v_{t_n}\right]^+\right) = g\left(t - u_t\right)$ by left-continuity) or a decreasing subsequence (then, $\lim_{n\to\infty} g\left(t_n - v_{t_n}\right) = g\left((t-u_t)^+\right)$ with $g(t^+) := \lim_{\varepsilon\searrow 0} g(t+\varepsilon)$). Either way, it holds that

$$\begin{aligned}
\lim_{n\to\infty} f \otimes g(t_n) &= \lim_{n\to\infty} F_{v_{t_n}}(t_n) \\
&= \lim_{n\to\infty} f(v_{t_n}) + g\left(t_n - v_{t_n}\right) \\
&\geq f(u_t) + g\left(t - u_t\right) \\
&= F_{u_t}(t) \\
&= f \otimes g(t),
\end{aligned}$$

where we used in the inequality that g is non-decreasing.

Moreover, by Lemma 2.3 in [BBL18, p. 22], the (min,+) convolution of non-decreasing functions is, again, non-decreasing, hence, $f \otimes g(t_n) \leq f \otimes g(t)$. Combining both inequalities finishes the proof. □

C.3 Attainability and left-continuity of (max,+) convolution

The proofs for Propositions 4.3 and 4.4 follow a similar approach.

Proposition 4.3 (Attainability of (max,+) Convolution)**.** *Let f and g be right-continuous, non-decreasing functions of $\mathcal{U}$. Then, for any $t \in \mathbb{Q}_+$ it exists $s^* \in [0,t]$ such that*

$$f \,\overline{\otimes}\, g(t) = \sup_{0\leq s\leq t} \{f(s) + g(t-s)\} = f(s^*) + g(t-s^*).$$

In other words, the supremum is always attained in $[0,t]$.

Proof. The proof follows by adapting the steps in the proof of Proposition 4.1 to the (max,+) convolution. Fix $t \geq 0$ and define $F_t(s) := f(s) + g(t-s)$. Let $(s_n)_{n\in\mathbb{N}}$ be a sequence such that $F_t(s_n)$ is increasing and converges to $\sup_{0\leq s\leq t}\{f(s) + g(t-s)\}$. Similar to the (min,+) convolution [BT08, Proposition 7], one can show that the limit, the (max,+) convolution $\sup_{0\leq s\leq t}\{f(s) + g(t-s)\}$, is in $\mathbb{Q}$. As for all n, it holds that $s_n \in [0,t]$. In other words, we have an infinite sequence on a closed and bounded interval. Therefore, by the Bolzano-Weierstrass theorem for real sequences, there exists a subsequence u_n that converges to $\bar{u} \in [0,t] \subset \mathbb{R}_+$. Moreover, by the Lemma C.1, this limit $\bar{u} \in \mathbb{Q}_+$. By the Monotone Subsequence Theorem, there exists a monotonic subsequence $(v_n)_{n\in\mathbb{N}}$ (of this subsequence u_n) with the same limit $\bar{u}$. Without loss of generality, this sequence is decreasing (if this sequence is increasing, we can exchange the role of f and g and replace v_n with $t - v_n$). As f is right-continuous, $\lim_{n\to\infty} f(v_n) = f(\bar{u})$, and, as g is non-decreasing, $\lim_{n\to\infty} g(t - v_n) \leq g(t - \bar{u})$. Therefore,

$$\lim_{n\to\infty} F_t(v_n) = \lim_{n\to\infty} f(v_n) + g(t - v_n) \leq f(\bar{u}) + g(t - \bar{u}) = F(\bar{u}).$$

Thus, F_t reaches its maximum on $[0,t]$. □

Proposition 4.4 (Right-Continuity of (max,+) Convolution)**.** *Let f and g be right-continuous, non-decreasing functions of $\mathcal{U}$. Then, $f \overline{\otimes} g$ is right-continuous.*

Proof. The proof is inspired by Proposition 3.11 in [BBL18, p. 49].

We extend the function g by

$$\tilde{g}(t) := \begin{cases} g(t), & \text{if } t \geq 0, \\ -\infty, & \text{otherwise.} \end{cases}$$

For all $t \geq 0$, it holds that $f \overline{\otimes} g(t) = f \overline{\otimes} \tilde{g}(t)$ and thus, $f \overline{\otimes} g(t)$ is right-continuous if and only if $f \overline{\otimes} \tilde{g}(t)$ is right-continuous for all $t \geq 0$.

For all $s \geq 0$, define $F_s : t \to f(s) + \tilde{g}(t - s)$. The right-continuity of F_s directly follows from that of $\tilde{g}$ and for all $t \geq 0$,

$$\begin{aligned} f \overline{\otimes} g(t) &= \sup_{0\leq s\leq t} \{F_s(t)\} \\ &= \sup_{s\geq 0} \{F_s(t)\}, \end{aligned}$$

where we used that the supremum is never attained for negative arguments of $\tilde{g}$. Note that from Proposition 4.3, this is a maximum, and we denote one value by $u_t \in [0,t]$ such that $f \overline{\otimes} g(t) = F_{u_t}(t)$.

Fix $t \geq 0$ and let t_n be a non-increasing sequence converging to t. To prove the right-continuity of $f \overline{\otimes} g(t)$, we need to show that $F_{u_n}(t_n)$ converges to $F_{u_t}(t)$, with $u_n = u_{t_n}$.

By the Monotone Subsequence Theorem, there exists a monotonic subsequence $(v_{t_n})_{n\in\mathbb{N}}$ (of u_n) with the same limit u_t. Without loss of generality, this sequence is decreasing (if this sequence is increasing, we can exchange the role of f and g and replace v_{t_n} with $t - v_{t_n}$). Since both sequences, t_n and v_{t_n}, are decreasing, we only know that $t_n - v_{t_n}$ converges to $t - u_t$, but not how. The sequence $\tilde{g}\left(t_n - v_{t_n}\right)$ either has a decreasing subsequence from above (in this case, $\lim_{n\to\infty} g\left(t_n - v_{t_n}\right) = g\left(t - u_t\right)$ by right-continuity) or an increasing subsequence with $t_n - v_{t_n} \geq 0$ from below (then, $\lim_{n\to\infty} g\left(t_n - v_{t_n}\right) = g\left(\left(t - u_t\right)^-\right)$ with $g(t^-) := \lim_{\varepsilon \searrow 0} g(t-\varepsilon)$). Either way, it holds that

$$\begin{aligned}
\lim_{n\to\infty} f \mathbin{\overline{\otimes}} g(t_n) &= \lim_{n\to\infty} F_{v_{t_n}}(t_n) \\
&= \lim_{n\to\infty} f(v_{t_n}) + g(t_n - u_{t_n}) \\
&\leq f(v_t) + g\left(t - u_t\right) \\
&= F_{u_t}(t) \\
&= f \mathbin{\overline{\otimes}} g(t),
\end{aligned}$$

where we used in the inequality that g is non-decreasing.

Moreover, by Lemma 4.1 in [Lie17, p. 17], the (max,+) convolution of non-decreasing functions is, again, non-decreasing, hence, $f \mathbin{\overline{\otimes}} g(t_n) \geq f \otimes g(t)$ (t_n is assumed to be a decreasing sequence). Combining both inequalities finishes the proof. □

C.4 Both functions must be *-continuous

[Lie17, p. 134] provides a result for the right-continuity of the (max,+) convolution that requires only one of the functions to be right-continuous. One may wonder if this results also applies to functions in $\mathcal{U}$.

The main step of [Lie17, p. 134], is the following equivalence, where the respective limit and supremum are exchanged:

$$\lim_{\varepsilon \searrow 0} \sup_{s\in\mathbb{R}} \left\{ f(s) + g\left(\left[t-s-\varepsilon\right]^+\right)\right\} = \sup_{s\in\mathbb{R}} \left\{ f(s) + g\left(\left[t-s\right]^+\right)\right\}.$$

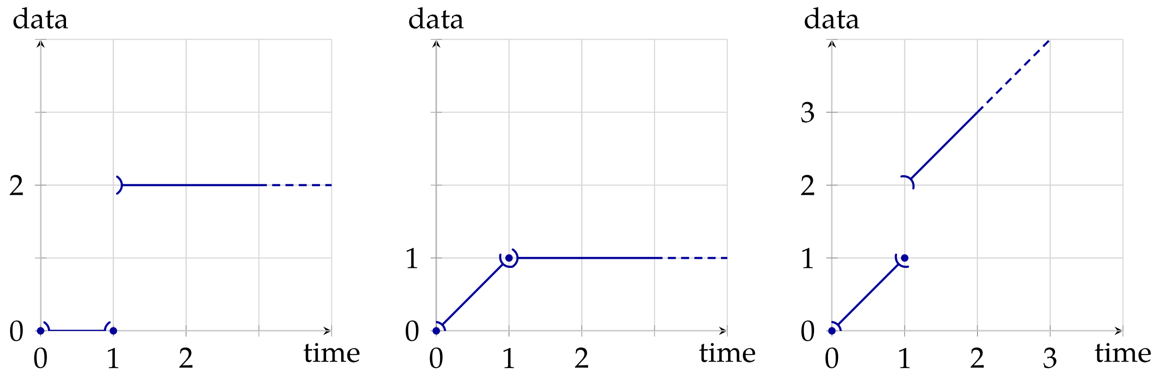
However, this step is, for functions of $\mathcal{U}$, generally an inequality "$\geq$". Consider the following example. Let

$$f(t) = \begin{cases} 0, & t \leq 1, \\ 2, & \text{else} \end{cases}$$

and

$$g(t) = \begin{cases} 0, & t < 0, \\ t, & 0 \leq t < 1, \\ 1, & \text{else.} \end{cases}$$

It is easy to check that f is not right-continuous (it is, in fact, left-continuous) and g is continuous; in particular, it is also right-continuous. Yet, their (max,+) convolution



(a) f, not right-continuous. (b) g, right-continuous. (c) $f \overline{\otimes} g$, not right-continuous.

Figure C.1: Example for the (max,+) convolution.

is not right-continuous anymore, as it holds that

$$
\begin{aligned}
f \overline{\otimes} g(1) = & \sup_{0 \le s \le 1} \left\{ \underbrace{f(s)}_{=0} + g(1-s) \right\} \\
= & g(1) = 1,
\end{aligned}
$$

while for any $\varepsilon > 0$, we have

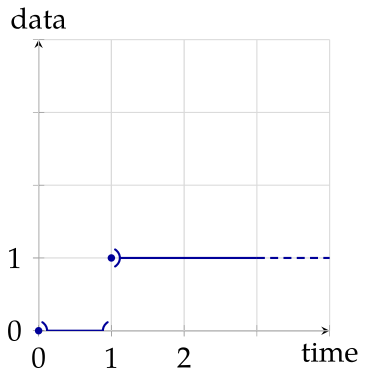
$$
\begin{aligned}
f \overline{\otimes} g(1+\varepsilon) &= \sup_{0 \le s \le 1+\varepsilon} \{ f(s) + g(1-s) \} \\
&= \sup_{1 < s \le 1+\varepsilon} \left\{ \underbrace{f(s)}_{=2} + g(1-s) \right\} = 2 + \varepsilon.
\end{aligned}
$$

The above is visualized in Figure C.1.

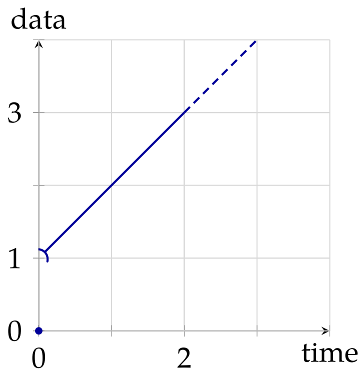
On the other hand, if one considers $t \in \mathbb{R}$ as in [Lie17], with $g(t) = 0$ for $t \le 0$, we obtain

$$
\begin{aligned}
f \overline{\otimes} g(1) = & \sup_{s \in \mathbb{R}} \{ f(s) + g(1-s) \} \\
= & f(2) + g(-1) = 2.
\end{aligned}
$$

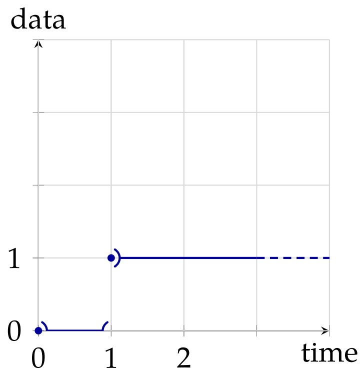
Thus, the result may apply only for convolutions of functions with unbounded support as discussed in [Lie17]. We can provide a similar example for the (min,+) convolution. This is given in Figure C.2.



(a) f, not left-continuous.



(b) g, left-continuous.



(c) $f \otimes g$, not left-continuous.

Figure C.2: Example for the $(\min,+)$ convolution.

Appendix D

Pseudoinverses

D.1 Differences in pseudoinverses definitions

In this section we discuss differences between the definition used in this work and [Lie17], and the impact on the results from that work that we use here. We note, also, that recently [PS22] provided a more general definition of pseudoinverses, which generalizes the sets of functions, properties such as Lemma 4.14, and does not require functions to be non-decreasing.

In [Lie17, p. 60], which considers functions from $\mathbb{R} \to \mathbb{R}$, lower and upper pseudoinverses are introduced as

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

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,

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

Proposition 4.10 states a weaker form of equivalence for functions in $\mathcal{U}$. We provide here a proof.

Proposition 4.10. *Let $f \in \mathcal{U}$ be non-decreasing. For all $y > f(0)$, its lower pseudoinverse is equal to*

$$f_{\downarrow}^{-1}(y) = \sup \{t \geq 0 \mid f(t) < y\}, \tag{4.4}$$

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

$$f_{\uparrow}^{-1}(y) = \inf \{t \geq 0 \mid f(t) > y\}. \tag{4.5}$$

Proof. The proof follows mostly along the lines of Lemma 3.2 in [BBL18, pp. 46].[1]

1. Lower pseudoinverse: 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 pseudoinverse: 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 $t_1 := \sup \{t \geq 0 \mid f(t) = f(0)\} \in \mathbb{Q}_+ \cup \{+\infty\}$. 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 $\{t \geq 0 \mid f(t) > f(0)\}$.

$\square$

Note that the discussion above applies to any bounded domain – see the more general discussion regarding pseudoinverses over a support, in Chapter 14. The same applies for the following property.

Lemma 4.14. *Let f be a non-decreasing function of $\mathcal{U}$. Then, for all $y \geq f(0)$,*

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

Proof. We distinguish cases. For $y > f(0)$, the claim follows immediately, since we can apply Proposition 4.10:

$$\begin{aligned} f_{\downarrow}^{-1}(y) &= \sup \{t \geq 0 \mid f(t) < y\} \\ &\leq \sup \{t \geq 0 \mid f(t) \leq y\} \\ &= f_{\uparrow}^{-1}(y). \end{aligned}$$

[1]We note however that Lemma 3.2 in [BBL18, pp. 46] is incomplete, since it does not account for the case $y \leq f(0)$ – see our counterexample above.

For $y = f(0)$, we calculate for the lower pseudoinverse

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

while the upper pseudoinverse yields

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

Combining both cases finishes the proof. □

Note that, for $y \in [0, f(0)[$, this is not the case. Then, we obtain for the lower pseudoinverse

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

while the upper pseudoinverse yields

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

which is obviously strictly less than $f_{\downarrow}^{-1}(y)$.

D.2 Properties of pseudoinverses

Lemma 4.11. *Let $f \in \mathcal{U}$ be non-decreasing. Then, $f_{\downarrow}^{-1}$ is left-continuous and $f_{\uparrow}^{-1}$ is right-continuous.*

Proof. The proof follows mostly along the lines of Lemma 10.1 in [Lie17, p. 65].

First, we prove the claim for the upper pseudoinverse. Let $y < f(0)$. Then,

$$\begin{aligned} f_{\uparrow}^{-1}\left(y^{+}\right) &= \lim_{\substack{x \searrow y,\\ x<f(0)}} \sup\{t \geq 0 \mid f(t) \leq x\} \\ &= \sup\{\emptyset\} \\ &= -\infty \\ &= \sup\{t \geq 0 \mid f(t) \leq y\} \\ &= f_{\uparrow}^{-1}(y). \end{aligned}$$

Now, let $y \geq f(0)$. Using Proposition 4.10, we derive

$$\begin{aligned} f_{\uparrow}^{-1}\left(y^{+}\right) &= \lim_{x \searrow y} \sup\{t \geq 0 \mid f(t) \leq x\} \\ &\overset{(4.5)}{=} \lim_{x \searrow y} \inf\{t \geq 0 \mid f(t) > x\} \\ &\overset{(f \text{ non-decr.})}{=} \inf_{x>y} \inf_{t}\{t \geq 0 \mid f(t) > x\} \\ &= \inf_{t,x}\{t \geq 0 \mid f(t) > x, x > y\} \\ &= \inf\{t \geq 0 \mid f(t) > y\} \\ &= f_{\uparrow}^{-1}(y). \end{aligned}$$

The first line expresses the limit from the right of the upper pseudoinverse for an arbitrary value $y \geq f(0)$. This finishes the proof for the upper pseudoinverse.

Let us now consider the lower pseudoinverse. Let $y \leq f(0)$. Then,

$$\begin{aligned}
f_{\downarrow}^{-1}\left(y^{+}\right) &= \lim_{x \nearrow y} \inf \{t \geq 0 \mid f(t) \geq x\} \\
&\overset{(x<f(0))}{=} 0 \\
&\overset{(y \leq f(0))}{=} \inf \{t \geq 0 \mid f(t) \geq y\} \\
&= f_{\downarrow}^{-1}(y) .
\end{aligned}$$

Last, let $y > f(0)$. Using Proposition 4.10, we derive

$$\begin{aligned}
f_{\downarrow}^{-1}\left(y^{+}\right) &= \lim_{x \nearrow y} \inf \{t \geq 0 \mid f(t) \geq x\} \\
&\overset{(4.4)}{=} \lim_{x \nearrow y} \sup \{t \geq 0 \mid f(t) < x\} \\
&\overset{(f \text{ non-decr.})}{=} \sup_{x<y} \sup_{t} \{t \geq 0 \mid f(t) < x\} \\
&= \sup_{t,x} \{t \geq 0 \mid f(t) < x, x < y\} \\
&= \sup \{t \geq 0 \mid f(t) < y\} \\
&= f_{\downarrow}^{-1}(y) .
\end{aligned}$$

□

Lemma 4.12. *Let $f \in \mathcal{U}$ be non-decreasing and left-continuous. Then,*

$$f = \left(f_{\uparrow}^{-1}\right)_{\downarrow}^{-1}. \tag{4.6}$$

Proof. We follow along the lines of [Lie17, p. 65], adapting the steps to consider $f \in \mathcal{U}$.

Let $t \in \mathbb{Q}_{+}$. Then

$$\begin{aligned}
\left(f_{\uparrow}^{-1}\right)_{\downarrow}^{-1}(t) &= \inf \left\{x \geq 0 \mid f_{\uparrow}^{-1}(x) \geq t\right\} \\
&= \inf \{x \geq 0 \mid \sup \{s \geq 0 \mid f(s) \leq x\} \geq t\} .
\end{aligned}$$

We introduce notation for the two sets appearing in the equation and define

$$\begin{aligned}
M_t &:= \{x \geq 0 \mid \sup \{s \geq 0 \mid f(s) \leq x\} \geq t\} , \\
M_{t,x} &:= \{s \geq 0 \mid f(s) \leq x\} .
\end{aligned}$$

We prove the claim by showing that $f(t) = \inf\{M_t\}$. Then, we show that $f(t) \in M_t$, and that if $\hat{x} < f(t)$, then $\hat{x} \notin M_t$. This would finish the proof as $f_\uparrow^{-1}$ is right-continuous and hence an infimum of the set $M_t = \left\{x \geq 0 \mid f_\uparrow^{-1}(x) \geq t\right\}$ must also be an element of M_t.

If $x = f(t)$, the condition to be an element of M_t becomes

$$\sup\{M_{t,x}\} = \sup\left\{M_{t,f(t)}\right\} = \sup\{s \geq 0 \mid f(s) \leq f(t)\} \geq t.$$

This supremum is either equal to t or equal to $s^* > t$ such that $f(s^*) = f(t)$. In both cases, the condition of the supremum to be greater than or equal to t is therefore satisfied, and hence we conclude that $f(t) \in M_t$.

For an $\hat{x} < f(t)$, the condition is equal to

$$\sup\{M_{t,\hat{x}}\} = \sup\{s \geq 0 \mid f(s) \leq \hat{x}\} \geq t.$$

Let $s^* := \sup\{M_{t,\hat{x}}\}$ be the supremum. Since f is left-continuous, we have that $s^* \in M_{t,\hat{x}}$, and, therefore $f(s^*) \leq \hat{x}$. Combining both assumptions yields

$$f(s^*) \leq \hat{x} < f(t).$$

Since f is non-decreasing, this implies that $s^* < t$. Hence, with $\hat{x}$, the condition for membership in M_t cannot be satisfied, and we have that $\hat{x} \notin M_t$. □

Lemma 4.13. *Let $f \in \mathcal{U}$ be non-decreasing and right-continuous. Then,*

$$f = \left(f_\downarrow^{-1}\right)_\uparrow^{-1}. \tag{4.7}$$

Proof. We follow along the lines of [Lie17, p. 65], adapting the steps to consider $f \in \mathcal{U}$.

Let $t \in \mathbb{Q}_+$. Then

$$\begin{aligned}\left(f_\downarrow^{-1}\right)_\uparrow^{-1}(t) &= \sup\left\{x \geq 0 \mid f_\downarrow^{-1}(x) \leq t\right\}\\ &= \sup\{x \geq 0 \mid \inf\{s \geq 0 \mid f(s) \geq x\} \leq t\}.\end{aligned}$$

We introduce notation for the two sets appearing in the equation and define

$$\begin{aligned}M_t &:= \{x \geq 0 \mid \inf\{s \geq 0 \mid f(s) \geq x\} \leq t\},\\ M_{t,x} &:= \{s \geq 0 \mid f(s) \geq x\}.\end{aligned}$$

We prove the claim by showing that $f(t) = \sup\{M_t\}$. Then, we show that $f(t) \in M_t$, and that if $\hat{x} > f(t)$, then $\hat{x} \notin M_t$. This would finish the proof as $f_\downarrow^{-1}$ is left-continuous and hence a supremum of the set $M_t = \left\{x \geq 0 \mid f_\downarrow^{-1}(x) \leq t\right\}$ must also be an element of M_t.

If $x = f(t)$, the condition to be an element of M_t becomes

$$\inf\{M_{t,x}\} = \inf\left\{M_{t,f(t)}\right\} = \inf\{s \geq 0 \mid f(s) \geq f(t)\} \leq t.$$

This infimum is either equal to t or equal to $0 \leq s^* < t$ such that $f(s^*) = f(t)$. In both cases, the condition of the infimum to be less than or equal to t is therefore satisfied, and hence we conclude that $f(t) \in M_t$.

For an $\hat{x} > f(t)$, the condition is equal to

$$\inf\{M_{t,\hat{x}}\} = \inf\{s \geq 0 \mid f(s) \geq \hat{x}\} \leq t.$$

Let $s^* := \inf\{M_{t,\hat{x}}\}$ be the infimum. Since f is right-continuous, we have that $s^* \in M_{t,\hat{x}}$, and, therefore $f(s^*) \geq \hat{x}$. Combining both assumptions yields

$$f(s^*) \geq \hat{x} > f(t).$$

Since f is non-decreasing, this implies that $s^* > t$. Hence, with $\hat{x}$, the condition for membership in M_t cannot be satisfied, and we have that $\hat{x} \notin M_t$. □

D.3 UPP properties of pseudoinverses

Lemma 10.3 (Sufficient Cut for Lower Pseudoinverse)**.** *Let f be a function of $\mathcal{U}$ that is non-decreasing and neither UC nor UI. Then, in order to compute $f_\downarrow^{-1}(x)$ with $x \in I' := [0, x'[$, where $x' \geq 0$, it is sufficient to use $f(t)$ with $t \in [0, t']$, where $t' := f_\downarrow^{-1}(x')$.*

Proof. By definition of

$$t' = f_\downarrow^{-1}(x') = \inf\left\{t \geq 0 \mid f(t) \geq x'\right\},$$

we can restrict the interval of the upper pseudoinverse as

$$\begin{aligned} f_\downarrow^{-1}(x') &= \inf\left\{t \geq 0 \mid f(t) \geq x'\right\} \\ &= \inf\left\{0 \leq t \leq t' \mid f(t) \geq x'\right\}. \end{aligned}$$

Since $f_\downarrow^{-1}$ is non-decreasing, $f_\downarrow^{-1}(0) \leq f_\downarrow^{-1}(x) \leq f_\downarrow^{-1}(x')$, so the restriction above can be applied for all $x \in I'$. Thus, it is sufficient to use only the values of f within interval $[0, t']$. □

Lemma 10.5 (Sufficient Cut for Upper Pseudoinverse)**.** *Let f be a function of $\mathcal{U}$ that is non-decreasing and neither UC nor UI. Then, in order to compute $f_\uparrow^{-1}(x)$ with $x \in I' := [0, x'[$, where $x' \geq 0$, it is sufficient to use $f(t)$ with $t \in [0, t']$, where $t' := f_\uparrow^{-1}(x')$.*

Proof. By definition of

$$t' = f_{\uparrow}^{-1}(x') = \sup\left\{t \geq 0 \mid f(t) \leq x'\right\},$$

we can restrict the interval of the upper pseudoinverse as

$$\begin{aligned} f_{\uparrow}^{-1}(x') &= \sup\left\{t \geq 0 \mid f(t) \leq x'\right\} \\ &= \sup\left\{0 \leq t \leq t' \mid f(t) \leq x'\right\}. \end{aligned}$$

Since $f_{\uparrow}^{-1}$ is non-decreasing, $f_{\uparrow}^{-1}(0) \leq f_{\uparrow}^{-1}(x) \leq f_{\uparrow}^{-1}(x')$, so the restriction above can be applied for all $x \in I'$. Thus, it is sufficient to use only the values of f within interval $[0, t']$. □

D.4 Calculation of lower and upper pseudoinverses

We report here the rigorous mathematical derivations for cases c1-c8 in Table 10.1.

Point after segment (cases c1-c4)

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

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

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

We then distinguish four cases based on two properties:

- Whether the segment is constant, i.e., $\rho = 0 \to b_1 = b_2$;
- Whether there is a discontinuity at t_2, i.e., $b_2 < b_3$.

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

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

and

$$f_{\uparrow}^{-1}(y) = \begin{cases} \sup\left\{x \mid f(x) \leq \overbrace{y}^{=b_1}\right\} = t_2, & \text{if } y = b_1 = f(t_1^+), \\ \sup\{x \mid f(x) \leq y\} = t_2, & \text{if } b_1 = f(t_1^+) < y < f(t_2), \\ \inf\left\{x \mid \underbrace{f(x)}_{=b_2} > y\right\} = \sup\{x \mid f(x) \leq y\} = t_2, & \text{if } y = f(t_2) = b_2. \end{cases} \tag{D.2}$$

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

$$f_{\downarrow}^{-1}(y) = \inf\left\{x \mid \overbrace{f(x)}^{=b_1} \geq y\right\} = t_1, \text{if } y = b_1. \tag{D.3}$$

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

$$f_{\uparrow}^{-1}(y) := \sup\left\{x \mid \underbrace{f(x)}_{=b_1} \leq y\right\} = t_2, \text{if } y = b_1. \tag{D.4}$$

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

$$f_{\downarrow}^{-1}(y) = \begin{cases} \inf\{x \mid f(x) \geq y\} = t_2, & \text{if } y = b_1 + r\,(t_2 - t_1) = f(t_2^-), \\ \inf\{x \mid f(x) \geq y\} = t_2, & \text{if } f(t_2^-) < y < f(t_2) = b_3, \\ \inf\left\{x \mid f(x) \geq \underbrace{y}_{=b_3}\right\} = t_2, & y = f(t_2) = b_3, \end{cases} \tag{D.5}$$

and

$$f_{\uparrow}^{-1}(y) = \begin{cases} \inf\{x \mid f(x) > y\} = t_2, & \text{if } y = b_1 + r\,(t_2 - t_1) = f(t_2^-), \\ \inf\{x \mid f(x) > y\} = t_2, & \text{if } f(t_2^-) < y < f(t_2) = b_3, \\ \inf\left\{x \mid f(x) > \underbrace{y}_{=b_3}\right\} = t_2, & y = f(t_2) = b_3. \end{cases} \tag{D.6}$$

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

$$f_{\downarrow}^{-1}(y) = \inf\left\{ x \mid f(x) \geq \underbrace{y}_{=b_2} \right\} = t_2,\ y = f(t_2) = b_2, \tag{D.7}$$

and

$$f_{\uparrow}^{-1}(y) = \inf\left\{ x \mid f(x) > \underbrace{y}_{=b_2} \right\} = t_2,\ y = f(t_2) = b_2. \tag{D.8}$$

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 the segment is constant, i.e., $\rho = 0 \to b_2 = b_3$;
- Whether 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

$$f_{\downarrow}^{-1}(y) = \begin{cases} \inf\{x \mid f(x) \geq y\} = \sup\{x \mid f(x) < y\} = t_1, & \text{if } b_1 < y < f(t_1^+) = b_2, \\ \inf\left\{ x \mid f(x) \geq \underbrace{y}_{=b_2} \right\} = t_1, & \text{if } y = f(t_1^+) = b_2, \end{cases} \tag{D.9}$$

and

$$f_{\uparrow}^{-1}(y) = \begin{cases} \inf\{x \mid f(x) > y\} = \sup\{x \mid f(x) \leq y\} = t_1, & \text{if } b_1 < y < f(t_1^+) = b_2, \\ \inf\{x \mid f(x) > y\} = \sup\left\{ x \mid f(x) \leq \underbrace{y}_{=b_2} \right\} = t_2, & \text{if } y = f(t_1^+) = b_2. \end{cases} \tag{D.10}$$

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

$$f_{\downarrow}^{-1}(y) = \inf\left\{ x \mid \overbrace{f(x)}^{=b_1} \geq y \right\} = t_1, \text{if } y = b_1. \tag{D.11}$$

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

$$f_{\uparrow}^{-1}(y) := \sup\left\{ x \mid \underbrace{f(x)}_{=b_1} \leq y \right\} = t_2, \text{if } y = b_1. \tag{D.12}$$

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

$$f_{\downarrow}^{-1}(y) = \begin{cases} \inf\{x \mid f(x) \geq y\} = t_1, & \text{if } b_1 < y < f(t_1^-) = b_2, \\ \inf\left\{ x \mid f(x) \geq \underbrace{y}_{=b_2} \right\} = t_1, & \text{if } y = f(t_1^-) = b_2, \\ \inf\{x \mid b_2 + \rho\,(x - t_1) \geq y\} = t_1 + \frac{y-b_2}{\rho}, & \text{if } b_2 < y < b_3, \end{cases} \tag{D.13}$$

and

$$f_{\uparrow}^{-1}(y) = \begin{cases} \inf\{x \mid f(x) > y\} = t_1, & \text{if } b_1 < y < b_2, \\ \inf\{x \mid f(x) > y\} = \sup\left\{ x \mid f(x) \leq \underbrace{y}_{=b_2} \right\} = t_1, & \text{if } y = b_2, \\ \inf\{x \mid b_2 + \rho\,(x - t_1) > y\} = t_1 + \frac{y-b_2}{\rho}, & \text{if } b_2 < y < b_3. \end{cases} \tag{D.14}$$

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

$$f_{\downarrow}^{-1}(y) = \inf\{x \mid b_1 + \rho\,(x - t_1) \geq y\} = t_1 + \frac{y - b_1}{\rho}, \quad \text{if } b_1 = f(t_1^+) < y < f(t_2^-) = b_2, \tag{D.15}$$

and

$$f_{\uparrow}^{-1}(y) = \inf\{x \mid b_1 + \rho\,(x - t_1) > y\} = t_1 + \frac{y - b_1}{\rho}, \quad \text{if } b_1 = f(t_1^+) < y < f(t_2^-) = b_2. \tag{D.16}$$

Appendix E

Composition

E.1 UPP properties of composition

Proposition 11.3. *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*

$$\begin{aligned} T_h &= \max\left(g_{\downarrow}^{-1}(T_f), T_g\right), \\ d_h &= \frac{d_f}{\rho_g}, && (11.5) \\ c_h &= c_f. && (11.6) \end{aligned}$$

Proof. Due to Theorem 11.1, $h \in \mathcal{U}$. Then, 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

$$\begin{aligned} h(t + k_h \cdot d_h) &= f\left(g(t + k_h \cdot d_h)\right) \\ &\overset{(3.2)}{=} f\left(g(T_h) + \rho_g \cdot (t + k_h \cdot d_h - T_h)\right) \\ &= f\left(\rho_g \cdot t + g(T_h) - \rho_g \cdot T_h + k_h \cdot d_f\right) \\ &\overset{(3.1)}{=} f\left(\rho_g \cdot t + g(T_h) - \rho_g \cdot T_h\right) + k_h \cdot c_f \\ &= f\left(g(T_h) + \rho_g \cdot (t - T_h)\right) + k_h \cdot c_f \\ &\overset{(3.2)}{=} f\left(g(t)\right) + k_h \cdot c_f \\ &= h(t) + k_h \cdot c_h. \end{aligned}$$

□

Proposition 11.4. *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*

$$\begin{aligned} T_h &= \max\left(g_{\downarrow}^{-1}(T_f), T_g\right), \\ d_h &= d_g, && (11.8) \\ c_h &= c_g \cdot \rho_f. && (11.9) \end{aligned}$$

Proof. Due to Theorem 11.1, $h \in \mathcal{U}$. Then, 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

$$\begin{aligned} h(t + k_h \cdot d_h) &= f\left(g(t + k_h \cdot d_h)\right) \\ &\overset{(3.1)}{=} f\left(g(t) + k_h \cdot c_g\right) \\ &\overset{(3.2)}{=} f(g(T_h)) + \rho_f \cdot \left(g(t) + k_h \cdot c_g - g(T_h)\right) \\ &= f(g(T_h)) + \rho_f \cdot (g(t) - g(T_h)) + k_h \cdot c_g \cdot \rho_f \\ &\overset{(3.2)}{=} f(g(t)) + k_h \cdot c_g \cdot \rho_f \\ &= h(t) + k_h \cdot c_h. \end{aligned}$$

□

Proposition 11.5. *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*

$$\begin{aligned} T_h^a &= \max\left(g_{\downarrow}^{-1}(T_f^a), T_g^a\right), \\ \rho_h &= \rho_f \cdot \rho_g. && (11.11) \end{aligned}$$

Proof. Due to Theorem 11.1, $h \in \mathcal{U}$. 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 $t \geq T_h^a$,

$$\begin{aligned} h(t + T_h^a) &= f\left(g(t + T_h^a)\right) \\ &\overset{(3.2)}{=} f\left(g(T_h^a) + \rho_g \cdot (t - T_h^a)\right) \\ &\overset{(3.2)}{=} f(g(T_h^a)) + \rho_f \cdot \left(\left(g(T_h^a) + \rho_g \cdot (t - T_h^a)\right) - g(T_h^a)\right) \\ &= f(g(T_h^a)) + \rho_f \cdot \rho_g \cdot (t - T_h^a) \\ &= h(T_h^a) + \rho_f \cdot \rho_g \cdot (t - T_h^a). \end{aligned}$$

□

E.2 Composition of Ultimately Constant (UC) functions

Proposition E.1. *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*

$$T_h = T_g. \tag{E.1}$$

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

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

□

Proposition E.2. *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*

$$T_h = g_{\downarrow}^{-1}(T_f). \tag{E.2}$$

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

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

□

Proposition E.3. *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*

$$T_h = \min\left(T_f, g_{\downarrow}^{-1}(T_f)\right). \tag{E.3}$$

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

Appendix F

Algorithmic Improvements through Isomorphism

F.1 Isomorphism of convolution for functions of $\mathcal{U}$

Theorem 4.16 (Isomorphism of Convolution For Left-Continuous Functions)**.** *Let f and g be functions of $\mathcal{U}$ that are left-continuous and non-decreasing. Then,*

$$(f \otimes g)_{\uparrow}^{-1} = \left(f_{\uparrow}^{-1}\right) \overline{\otimes} \left(g_{\uparrow}^{-1}\right). \tag{4.12}$$

Proof. The proof follows mostly along the lines of [Lie17, Theorem 10.3b]. The crucial difference is that functions of $\mathcal{U}$ have 0 as a boundary, while functions in [Lie17] are defined over all real numbers (see Section 4.3).

First, we consider the case $x < f(0) + g(0)$. We calculate

$$f \otimes g(0) = f(0) + g(0) > x.$$

Therefore, by the definition of the upper pseudoinverse, we know for all $x < f(0) + g(0)$ that

$$(f \otimes g)_{\uparrow}^{-1}(x) = \sup \{t \geq 0 \mid f \otimes g(t) \leq x\} = -\infty.$$

Similarly, we obtain

$$\left(f_{\uparrow}^{-1} \overline{\otimes} g_{\uparrow}^{-1}\right)(y) = -\infty$$

for all $y < f(0) + g(0)$. In fact, for any $y = v + (y - v)$, $0 \leq v \leq y$, either $v < f(0)$, thus $f_{\uparrow}^{-1}(v) = \sup \{t \geq 0 \mid f(t) \leq v\} = -\infty$, or $y - v < g(0)$, thus $g_{\uparrow}^{-1}(y - v) = \sup \{t \geq 0 \mid g(t) \leq y - v\} = -\infty$.

Now, let $x \geq f(0) + g(0)$. In this case, x is in the interval $[f(0) + g(0), +\infty[$, hence we can follow the exact same steps of [Lie17, Theorem 10.3b, p. 69] to obtain

$$(f \otimes g)_{\uparrow}^{-1} = \left(f_{\uparrow}^{-1}\right) \overline{\otimes} \left(g_{\uparrow}^{-1}\right).$$

□

Theorem 4.18 (Isomorphism of Convolution For Right-Continuous Functions)**.** *Let f and g be functions of $\mathcal{U}$ that are right-continuous and non-decreasing. Then,*

$$(f \overline{\otimes} g)_{\downarrow}^{-1} = \left(f_{\downarrow}^{-1}\right) \otimes \left(g_{\downarrow}^{-1}\right). \tag{4.14}$$

Proof. The proof follows mostly along the lines of Theorem 4.16, once adapted for (max,+) algebra. Again, the crucial difference is that functions of $\mathcal{U}$ have 0 as a boundary, while functions in [Lie17] are defined over all real numbers (see Section 4.3).

First, we consider the case $x \leq f(0) + g(0)$. We calculate

$$(f \overline{\otimes} g)\,(0) = f(0) + g(0) \geq x.$$

Therefore, by the definition of the lower pseudoinverse, we know for all $x \leq f(0) + g(0)$ that

$$(f \overline{\otimes} g)_{\downarrow}^{-1}(x) = \inf\{t \geq 0 \mid f(t) \geq x\} = 0.$$

Similarly, we obtain

$$\left(f_{\downarrow}^{-1} \otimes g_{\downarrow}^{-1}\right)(y) = 0$$

for all $y \leq f(0) + g(0)$, since we know that $f_{\downarrow}^{-1}(y) = \inf\{t \geq 0 \mid f(t) \geq y\} = 0$ for $y \leq f(0)$ and $g_{\uparrow}^{-1}(y) = \inf\{t \geq 0 \mid g(t) \geq y\} = 0$ for $y \leq g(0)$, respectively.

Now, let $x > f(0) + g(0)$. In this case, x is an interior point in the interval $[f(0) + g(0), +\infty[$, hence we can follow the exact same steps of [Lie17, Theorem 10.4b, p. 69][1] to obtain

$$(f \overline{\otimes} g)_{\downarrow}^{-1} = \left(f_{\downarrow}^{-1}\right) \otimes \left(g_{\downarrow}^{-1}\right).$$

□

F.2 Isomorphism of convolution for restricted functions

Lemma 14.11 (Improved period start for Lower Pseudoinverse over Interval)**.** *Let $f \in \mathcal{U}$ be neither UC nor UI, right-continuous, and non-decreasing over the interval $I = [a, +\infty[$, where $a \leq T_f$. Let*

$$\begin{aligned} T_f^* &:= f_{\downarrow,I}^{-1}(f(T_f)) \\ &= \inf\left\{t \geq a \mid f(t) \geq f(T_f)\right\} \\ &= \inf\left\{t \geq a \mid f(t) = f(T_f)\right\}, \end{aligned} \tag{14.15}$$

[1]Note that [Lie17] does not provide an explicit proof for Theorem 10.4, as it requires only minor changes from that of Theorem 10.3.

and

$$\begin{aligned} T_f^{**} &:= f_{\downarrow,I}^{-1}(f(T_1^* + d_f)) \\ &= \inf\left\{t \geq a \mid f(t) = f(T_1^* + d_f)\right\}. \end{aligned} \tag{14.16}$$

Then, if f is UPP from T_f^* and if $T_f^{**} = T_f^* + d_f$, the pseudo-periodic start $T_{f_\uparrow^{-1}}$ in Equation (14.9) *can be improved into*

$$T_{f_{\downarrow,I}^{-1}} = f(T_f). \tag{14.17}$$

Proof. Since f is right-continuous over I, it follows that $f(T_f^*) = f(T_f)$ and $f(T_f^{**}) = f(T_f^* + d_f)$.

Let $t_1 \in \left[T_f^*, T_f^* + d_f\right[$ and $x := f(t_1)$. Moreover, we define

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

By definition of T_f^*, $T_f^* \leq t_0 \leq t_1$. Moreover, from $T_f^{**} = \inf\left\{t \geq a \mid f(t) = f(T_f^* + d_f)\right\} = T_f^* + d_f$ it follows that $f(\tau) < f(T_f^* + d_f)$ for any $\tau < T_1^* + d_f$, and subsequently, it holds that $f(t_0) < f(T_f^* + d_f) \leq f(t_0 + d_f)$. Therefore,

$$\begin{aligned} f_{\downarrow,I}^{-1}\left(x + d_{f_{\downarrow,I}^{-1}}\right) &= \inf\left\{t \geq a \mid f(t) \geq x + d_{f_{\downarrow,I}^{-1}}\right\} \\ &\overset{(14.10)}{=} \inf\left\{t \geq a \mid f(t) \geq x + c_f\right\} \\ &= \inf\left\{t \geq a \mid f(t) \geq f(t_1) + c_f\right\} \\ &= \inf\left\{t \geq a \mid f(t) \geq f(t_0) + c_f\right\} \\ &= \inf\left\{t \geq a \mid f(t) \geq f(t_0 + d_f)\right\} \\ &= t_0 + d_f \\ &\overset{(14.11)}{=} f_{\downarrow,I}^{-1}(x) + c_{f_{\downarrow,I}^{-1}}, \end{aligned}$$

where we used that $a \leq T_f^* \leq t_0 + d_f$ on the second-to-last line.

Then, for any $k \in \mathbb{N}$, since $x + k \cdot c_f$ is again $\geq f(T_f^*)$, it follows by induction that

$$f_{\downarrow,I}^{-1}\left(x + k \cdot c_f\right) = f_{\downarrow,I}^{-1}\left(x + c_f\right) + k \cdot d_f,$$

thus satisfying the UPP property (Equation (3.1)).

This concludes the proof. □

Lemma 14.13. *Let $f \in \mathcal{U}$ be neither UC nor UI, left-continuous, and non-decreasing over the interval $I = [a, +\infty[$, where $a \leq T_f$. Moreover, let $f_{\uparrow,I}^{-1}$ be its upper pseudoinverse over I. Then, $f_{\uparrow,I}^{-1}$ satisfies the conditions of Lemma 14.11. Thus, its lower pseudoinverse $\left(f_{\uparrow,I}^{-1}\right)_{\downarrow,[f(a),+\infty[}^{-1}$ is UPP from $f_{\uparrow,I}^{-1}(T_{f_{\uparrow,I}^{-1}}) = T_f$.*

Proof. By Theorem 14.10, $f^{-1}_{\uparrow,I}$ is UPP from $f(T_f)$. What is left to show, is that $T^*_{f^{-1}_{\uparrow,I}} = f(T_f)$ as well as $T^{**}_{f^{-1}_{\uparrow,I}} = T^*_{f^{-1}_{\uparrow,I}} + d_{f^{-1}_{\uparrow,I}}$. As discussed in Chapter 10, a constant segment in $f^{-1}_{\uparrow,I}$ corresponds to a jump, i.e., a discontinuity, in f. In the following, we show that, due to left-continuity, there are no jumps in f such that $f^{-1}_{\uparrow,I}$ may have constant segments that would cause $T^{**}_f < T^*_{f^{-1}_{\uparrow,I}} + d_{f^{-1}_{\uparrow,I}}$ which would lead to a contradiction. In particular this holds at T_f and $T_f + d_f$.

If $T_f = a$, then

$$T^*_{f^{-1}_{\uparrow,I}} \overset{(14.15)}{=} \inf\left\{y \geq f(a) \mid f^{-1}_{\uparrow,I}(y) = f^{-1}_{\uparrow,I}(f(a))\right\} = f(a) = f(T_f). \tag{F.1}$$

Now, let $\tau > a$, and let $x := f(\tau)$. Assume there exists $x^* < x$ such that $f^{-1}_{\uparrow,I}$ is constant in $[x^*, x]$. Therefore,

$$\begin{aligned}\sup\{t \geq a \mid f(t) \leq x\} &= f^{-1}_{\uparrow,I}(x)\\ &= f^{-1}_{\uparrow,I}(x^*)\\ &= \sup\{t \geq a \mid f(t) \leq x^*\}.\end{aligned}$$

In order to find the supremum t such that $f(t) \leq x$, it is sufficient to consider only t such that $f(t) \leq x^*$; or, in other words, the function f jumps between x^* and $x = f(\tau)$, which is a contradiction to f being left-continuous in τ. Indeed, for $\varepsilon \in \left]0, x - x^*\right[$, there exists no $\delta > 0$ such that $\tau - \delta < t < \tau$ implies $|f(t) - f(\tau)| = |f(t) - x| < \varepsilon$. Therefore, for above-mentioned $x = f(\tau)$, $f^{-1}_{\uparrow,I}$ will not have a constant segment on $\left]x - \varepsilon, x\right]$ for any $\varepsilon > 0$.

Since this was true for all $\tau > a$, in particular, it is true for $T_f > a$ and $T_f + d_f$, implying that

$$T^*_{f^{-1}_{\uparrow,I}} \overset{(14.15)}{=} \inf\left\{y \geq f(a) \mid f^{-1}_{\uparrow,I}(y) = f^{-1}_{\uparrow,I}(f(T_f))\right\} = f(T_f)$$

and

$$T^{**}_{f^{-1}_{\uparrow,I}} \overset{(14.16)}{=} \inf\left\{y \geq f(a) \mid f^{-1}_{\uparrow,I}(y) = f^{-1}_{\uparrow,I}\left(T^*_{f^{-1}_{\uparrow,I}} + d_{f^{-1}_{\uparrow,I}}\right)\right\} = T^*_{f^{-1}_{\uparrow,I}} + d_{f^{-1}_{\uparrow,I}}.$$

Combining this with Equation (F.1) finishes the proof. □

Lemma 14.14 (Sufficient Cut for Lower Pseudoinverse over Interval)**.** *Let $f \in \mathcal{U}$ be neither UC nor UI, and is right-continuous and non-decreasing over $I = \left[a, +\infty\right[$. Then, in order to compute $f^{-1}_{\downarrow,I}(x)$ with $x \in [x_1, x_2] \subset \left[f(a), +\infty\right[$ and $x_1 < x_2$, it is sufficient to use $f(t)$ with $t \in [t_1, t_2]$, where $t_1 := f^{-1}_{\downarrow,I}(x_1)$ and $t_2 := f^{-1}_{\downarrow,I}(x_2)$.*

Proof. Let $x \in [x_1, x_2] \subset [f(a), +\infty[$ with $x_1 < x_2$. Since $f^{-1}_{\downarrow,I}$ is non-decreasing, $f^{-1}_{\downarrow,I}(x_1) \leq f^{-1}_{\downarrow,I}(x) \leq f^{-1}_{\downarrow,I}(x_2)$. Thus, we can focus on the boundaries. We obtain By definition of

$$t_1 = f^{-1}_{\downarrow,I}(x_1) = \inf\{t \geq a \mid f(t) \geq x_1\},$$

it follows that $f(t_1) \geq x_1$ (by right-continuity of f) and $f(t_1 - \varepsilon) < x_1$ for any $\varepsilon \geq 0$ such that $t_1 - \varepsilon \geq a$. Thus,

$$\begin{aligned} f^{-1}_{\downarrow,I}(x_1) &= \inf\{t \geq a \mid f(t) \geq x_1\} \\ &= \inf\{t \geq t_1 \mid f(t) \geq x_1\} \\ &= f^{-1}_{\downarrow,[t_1,+\infty[}(x_1). \end{aligned}$$

Moreover, by definition of

$$t_2 = f^{-1}_{\downarrow,I}(x_2) = \inf\{t \geq a \mid f(t) \geq x_2\},$$

we can restrict the interval of the upper pseudoinverse as

$$\begin{aligned} f^{-1}_{\downarrow,I}(x_2) &= \inf\{t \geq a \mid f(t) \geq x_2\} \\ &= \inf\{a \leq t \leq t_2 \mid f(t) \geq x_2\} \\ &= f^{-1}_{\downarrow,[a,t_2]}(x_2), \end{aligned}$$

where we used the right-continuity of f in the second line. Combining the two results, for any $x \in [x_1, x_2]$ we obtain

$$\begin{aligned} f^{-1}_{\downarrow,I}(x) &= \inf\{t \geq a \mid f(t) \geq x\} \\ &= \inf\{t_1 \leq t \leq t_2 \mid f(t) \geq x\} \\ &= f^{-1}_{\downarrow,[t_1,t_2]}(x) \end{aligned}$$

Thus only the values of f within said interval are required. □

Lemma 14.15 (Sufficient Cut for Upper Pseudoinverse over Interval)**.** *Let $f \in \mathcal{U}$ be neither UC nor UI, and is left-continuous and non-decreasing over $I = [a, +\infty[$. Then, in order to compute $f^{-1}_{\uparrow,I}(x)$ and $x \in [x_1, x_2] \subset [f(a), +\infty[$ with $x_1 < x_2$, it is sufficient to use $f(t)$ with $t \in [t_1, t_2]$, where $t_1 := f^{-1}_{\uparrow,I}(x_1)$ and $t_2 := f^{-1}_{\uparrow,I}(x_2)$.*

Proof. Let $x \in [x_1, x_2] \subset [f(a), +\infty[$ with $x_1 < x_2$. Since $f^{-1}_{\uparrow,I}$ is non-decreasing, $f^{-1}_{\uparrow,I}(x_1) \leq f^{-1}_{\uparrow,I}(x) \leq f^{-1}_{\uparrow,I}(x_2)$. Thus, we can focus on the boundaries. By definition of

$$t_1 := f^{-1}_{\uparrow,I}(x_1) = \sup\{t \geq a \mid f(t) \leq x_1\},$$

it follows that $f(t_1) \leq x_1$ (by left-continuity of f) and $f(t_1 + \varepsilon) > x_1$ for any $\varepsilon > 0$. Thus,

$$\begin{aligned} f^{-1}_{\uparrow,I}(x_1) &= \sup \{t \geq a \mid f(t) \leq x_1\} \\ &= \sup \{t \geq t_1 \mid f(t) \leq x_1\} \\ &= f^{-1}_{\uparrow,[t_1,+\infty[}(x_1). \end{aligned}$$

Moreover, by definition of

$$t_2 := f^{-1}_{\uparrow,I}(x_2) = \sup \{t \geq a \mid f(t) \leq x_2\},$$

we can restrict the interval of the upper pseudoinverse as

$$\begin{aligned} f^{-1}_{\uparrow,I}(x_2) &= \sup \{t \geq a \mid f(t) \leq x_2\} \\ &= \sup \{a \leq t \leq t_2 \mid f(t) \leq x_2\} \\ &= f^{-1}_{\uparrow,[a,t_2]}(x_2), \end{aligned}$$

where we used the left-continuity of f in the second line. Combining the two results, for any $x \in [x_1, x_2]$ we obtain

$$\begin{aligned} f^{-1}_{\uparrow,I}(x) &= \sup \{t \geq a \mid f(t) \leq x\} \\ &= \sup \{t_1 \leq t \leq t_2 \mid f(t) \leq x\} \\ &= f^{-1}_{\uparrow,[t_1,t_2]}(x) \end{aligned}$$

Thus, only the values of f within said interval are required. □

F.2.1 Isomorphism of restricted (min,+) convolution

Lemma 14.16. *Let $f \in \mathcal{U}$ be non-decreasing over $I = [a, +\infty[\subset \mathbb{Q}_+$. Let $x \in I$. If $f(x) \leq y$, then $f^{-1}_{\uparrow,I}(y) \geq x$.*

Proof. The proof follows mostly along the lines of [Lie17, p. 62]. Since $x \in I$ and $f(x) \leq y$, applying $f^{-1}_{\uparrow,I}$ to $f(x) \leq y$ preserves the inequality (we are in the supremum case of Equation (14.6)) and thus it follows that

$$f^{-1}_{\uparrow,I}(f(x)) \leq f^{-1}_{\uparrow,I}(y).$$

Moreover, since $x \in I$, it holds that

$$\begin{aligned} f^{-1}_{\uparrow,I}(f(x)) &= \sup \{z \in I \mid f(z) \leq f(x)\} \\ &\geq x. \end{aligned}$$

This finishes the proof. □

Proposition 14.17. *Let $f_p^\wedge, g_p^\wedge \in \mathcal{U}$ be left-continuous and non-decreasing, respectively, over $[T_f, +\infty[$ and $[T_g, +\infty[$. Then, for any $t \in [T_f + T_g, +\infty[$ it exists $s^* \in [T_f, t - T_g]$ such that*

$$\begin{aligned}(f_p^\wedge \otimes g_p^\wedge)(t) &= \inf_{0 \le s \le t} \left\{ f_p^\wedge(s) + g_p^\wedge(t-s) \right\} \\ &= \inf_{T_f \le s \le t - T_g} \left\{ f_p^\wedge(s) + g_p^\wedge(t-s) \right\} \\ &= f_p^\wedge(s^*) + g_p^\wedge(t-s^*).\end{aligned}$$

In other words, the infimum is attainable.

Proof. The proof follows along the lines of [BBL18, pp. 48].

First, we justify the first equivalence, i.e.,

$$\inf_{0 \le s \le t} \left\{ f_p^\wedge(s) + g_p^\wedge(t-s) \right\} = \inf_{T_f \le s \le t - T_g} \left\{ f_p^\wedge(s) + g_p^\wedge(t-s) \right\}$$

If $s < T_f$, then $f_p^\wedge(s) = +\infty$, and similarly if $s > t - T_g$, then $t - s < T_g$ and $g_p^\wedge(t-s) = +\infty$. Thus both cases can be excluded from the infimum.

Fix $t \ge T_f + T_g$ and define $F_t(s) := f_p^\wedge(s) + g_p^\wedge(t-s)$. Let $(s_n)_{n \in \mathbb{N}}$ be a sequence such that $F_t(s_n)$ is decreasing and converges to

$$\inf_{T_f \le s \le t - T_g} \left\{ f_p^\wedge(s) + g_p^\wedge(t-s) \right\}. \tag{F.2}$$

Note that (F.2) is in $\mathbb{Q}$, as it is true for (min,+) convolution of two piecewise affine functions in general [BT08, Proposition 7]. As for all n, it holds that $s_n \in [T_f, t - T_g]$. In other words, we have an infinite sequence on a closed and bounded interval. Therefore, by the Bolzano-Weierstrass theorem for real sequences, there exists a subsequence u_n that converges to $\bar{u} \in [T_f, t - T_g] \subset \mathbb{R}_+$. Moreover, by the Lemma C.1, this limit $\bar{u} \in \mathbb{Q}_+$. By the Monotone Subsequence Theorem, there exists a monotonic subsequence $(v_n)_{n \in \mathbb{N}}$ (of this subsequence u_n) with the same limit $\bar{u}$. Without loss of generality, this sequence is increasing (if this sequence is decreasing, we can exchange the role of $f_p^\wedge$ and $g_p^\wedge$ and replace v_n with $t - v_n$). As $f_p^\wedge$ is left-continuous over $[T_f, +\infty[$, $\lim_{n\to\infty} f_p^\wedge(v_n) = f_p^\wedge(\bar{v})$, and, as $g_p^\wedge$ is non-decreasing over $[T_g, +\infty[$, $\lim_{n\to\infty} g_p^\wedge(t - v_n) \ge g_p^\wedge(t - \bar{v})$. Therefore,

$$\lim_{n\to\infty} F_t(v_n) = \lim_{n\to\infty} f_p^\wedge(v_n) + g_p^\wedge(t - v_n) \ge f_p^\wedge(\bar{u}) + g_p^\wedge(t - \bar{u}) = F(\bar{u}).$$

Thus, F_t reaches its minimum on $[T_f, t - T_g] \subseteq [0, t]$. □

Theorem 14.18. *Let $f_p^\wedge, g_p^\wedge \in \mathcal{U}$ be left-continuous and non-decreasing, respectively, over $[T_f, +\infty[$ and $[T_g, +\infty[$. Then*

$$\left(f_p^\wedge \otimes g_p^\wedge\right)^{-1}_{\uparrow, [T_f + T_g, +\infty[} = \left(f_{\uparrow,p}^{-1} \overline{\otimes} g_{\uparrow,p}^{-1}\right). \tag{14.18}$$

Proof. The proof follows mostly along the lines of [Lie17, Theorem 10.3, p. 69].

First, we consider the case $x < f(T_f) + g(T_g)$. We calculate

$$\left(f_p^{\wedge} \otimes g_p^{\wedge}\right)(T_f + T_g) = f(T_f) + g(T_g) > x,$$

where we used for the equality that $f_p^{\wedge}(t) = +\infty$ for $t < T_f$ and $g_p^{\wedge}(t) = +\infty$ for $t < T_g$. Therefore, by the definition of the upper pseudoinverse over an interval, we know for all $x < f(T_f) + g(T_g)$ that

$$\left(f_p^{\wedge} \otimes g_p^{\wedge}\right)^{-1}_{\uparrow, \left[T_f + T_g, +\infty\right[}(x) = -\infty.$$

Similarly, we obtain

$$\left(f^{-1}_{\uparrow,p} \overline{\otimes} g^{-1}_{\uparrow,p}\right)(y) = -\infty$$

for all $y < f(T_f) + g(T_g)$, since we know that $f^{-1}_{\uparrow,p}(y) = -\infty$ for $y < f(T_f)$ and $g^{-1}_{\uparrow,p}(y) = -\infty$ for $y < g(T_g)$, respectively.

Now, let $x \geq f(T_f) + g(T_g)$. First, we show that

$$\left(f_p^{\wedge} \otimes g_p^{\wedge}\right)^{-1}_{\uparrow, \left[T_f + T_g, +\infty\right[}(x) \leq \left(f^{-1}_{\uparrow,p} \overline{\otimes} g^{-1}_{\uparrow,p}\right)(x).$$

Define

$$s_t^* := \operatorname*{arg\,inf}_{T_f \leq s \leq t - T_g} \{f_p^{\wedge}(s) + g_p^{\wedge}(t-s)\}$$

and $y_t^* := f_p^{\wedge}(s_t^*)$. Note that, by Proposition 14.17, s_t^* exists and is in $[T_f, t - T_g]$ since $f_p^{\wedge}$ and $g_p^{\wedge}$ are left-continuous over $\left[T_f, +\infty\right[$ and $\left[T_g, +\infty\right[$, respectively. We continue with

$$\begin{aligned}
&\left(f_p^{\wedge} \otimes g_p^{\wedge}\right)^{-1}_{\uparrow, \left[T_f + T_g, +\infty\right[}(x) \\
&= \sup\left\{t \in \left[T_f + T_g, +\infty\right[\mid \left(f_p^{\wedge} \otimes g_p^{\wedge}\right)(t) \leq x\right\} \\
&= \sup\left\{t \in \left[T_f + T_g, +\infty\right[\mid \inf_{0 \leq s \leq t}\left\{f_p^{\wedge}(s) + g_p^{\wedge}(t-s)\right\} \leq x\right\} \\
&= \sup\left\{t \in \left[T_f + T_g, +\infty\right[\mid \inf_{T_f \leq s \leq t - T_g}\left\{f_p^{\wedge}(s) + g_p^{\wedge}(t-s)\right\} \leq x\right\} \\
&= \sup\left\{t \in \left[T_f + T_g, +\infty\right[\mid \underbrace{f_p^{\wedge}(s_t^*)}_{=y_t^*} + g_p^{\wedge}(t - s_t^*) \leq x\right\}
\end{aligned}$$

$$\begin{aligned}
&= \sup \left\{ t \in \left[T_f + T_g, +\infty \right[\mid f_p^{\wedge}(s_t^*) \leq y_t^* \text{ and } g_p^{\wedge}(t - s_t^*) \leq x - y_t^* \right\} \\
&\overset{\text{(Lemma 14.16)}}{\leq} \sup \left\{ t \in \left[T_f + T_g, +\infty \right[\mid f_{\uparrow,p}^{-1}(y_t^*) \geq s_t^* \text{ and } g_{\uparrow,p}^{-1}(x - y_t^*) \geq t - s_t^* \right\} \\
&\leq \sup \left\{ t \in \left[T_f + T_g, +\infty \right[\mid f_{\uparrow,p}^{-1}(y_t^*) + g_{\uparrow,p}^{-1}(x - y_t^*) \geq t \right\} \\
&\leq \sup \left\{ t \in \left[T_f + T_g, +\infty \right[\mid \sup_{0 \leq z \leq x} \left\{ f_{\uparrow,p}^{-1}(z) + g_{\uparrow,p}^{-1}(x - z) \right\} \geq t \right\} \\
&= \sup \left\{ t \in \left[T_f + T_g, +\infty \right[\mid \left(f_{\uparrow,p}^{-1} \overline{\otimes} g_{\uparrow,p}^{-1} \right)(x) \geq t \right\} \\
&\leq \left(f_{\uparrow,p}^{-1} \overline{\otimes} g_{\uparrow,p}^{-1} \right)(x)
\end{aligned}$$

where we used again that $\left(f_{\uparrow,p}^{-1} \overline{\otimes} g_{\uparrow,p}^{-1} \right)(x) \geq T_f + T_g$ for all $x \geq f(T_f) + g(T_g)$. Moreover, we used that

$$\left(f_{\uparrow,p}^{-1}(y_t^*) \geq s_t^* \text{ and } g_{\uparrow,p}^{-1}(x - y_t^*) \geq t - s_t^* \right) \Rightarrow g_{\uparrow,p}^{-1}(x - y_t^*) + f_{\uparrow,p}^{-1}(y_t^*) \geq t$$

when replacing the "and" by the sum. For more details about this step, see Appendix G.3.

For the reverse direction, we derive

$$\begin{aligned}
&\left(f_{\uparrow,p}^{-1} \overline{\otimes} g_{\uparrow,p}^{-1} \right)(x) \\
&= \sup_{0 \leq z \leq x} \left\{ f_{\uparrow,p}^{-1}(z) + g_{\uparrow,p}^{-1}(z - x) \right\} \\
&= \sup_{f(T_f) \leq z \leq x - g(T_g)} \left\{ f_{\uparrow,p}^{-1}(z) + g_{\uparrow,p}^{-1}(z - x) \right\} \\
&= \sup_{f(T_f) \leq z \leq x - g(T_g)} \left\{ \sup \left\{ \tau \in \left[T_f, +\infty \right[\mid f(\tau) \leq z \right\} + \sup \left\{ s \in \left[T_g, +\infty \right[\mid g(s) \leq x - z \right\} \right\} \\
&= \sup_{f(T_f) \leq z \leq x - g(T_g)} \left\{ \sup \left\{ \tau \in \left[T_f, +\infty \right[\mid f_p^{\wedge}(\tau) \leq z \right\} + \sup \left\{ s \in \left[T_g, +\infty \right[\mid g_p^{\wedge}(s) \leq x - z \right\} \right\} \\
&= \sup_{f(T_f) \leq z \leq x - g(T_g)} \left\{ \sup \left\{ t \in \left[T_f + T_g, +\infty \right[\mid t = \tau + s \text{ and } f_p^{\wedge}(\tau) \leq z \text{ and } g_p^{\wedge}(s) \leq x - z \right\} \right\} \\
&\leq \sup_{f(T_f) \leq z \leq x - g(T_g)} \left\{ \sup \left\{ t \in \left[T_f + T_g, +\infty \right[\mid t = \tau + s \text{ and } f_p^{\wedge}(\tau) + g_p^{\wedge}(s) \leq x \right\} \right\} \\
&= \sup_{f(T_f) \leq z \leq x - g(T_g)} \left\{ \sup \left\{ t \in \left[T_f + T_g, +\infty \right[\mid \inf_{0 \leq \tau \leq t} \left\{ f_p^{\wedge}(\tau) + g_p^{\wedge}(t - \tau) \right\} \leq x \right\} \right\} \\
&= \sup_{f(T_f) \leq z \leq x - g(T_g)} \left\{ \sup \left\{ t \in \left[T_f + T_g, +\infty \right[\mid \left(f_p^{\wedge} \otimes g_p^{\wedge} \right)(t) \leq x \right\} \right\} \\
&= \sup \left\{ t \in \left[T_f + T_g, +\infty \right[\mid \left(f_p^{\wedge} \otimes g_p^{\wedge} \right)(t) \leq x \right\}
\end{aligned}$$

$$=\left(f_p^\wedge \otimes g_p^\wedge\right)^{-1}_{\uparrow,\left[T_f+T_g,+\infty\right[}(x),$$

where, again, we used that $\left(f_p^\wedge \otimes g_p^\wedge\right)^{-1}_{\uparrow,\left[T_f+T_g,+\infty\right[}(x) \geq T_f + T_g$ for $x \geq f(T_f) + g(T_g)$. This finishes the proof. □

Proposition 14.19. *Let $f \in \mathcal{U}$ be left-continuous and non-decreasing on the interval $I = [a, +\infty[$. Let $a' := \sup\{t \geq a \mid f(t) = f(a)\} \geq a$. Then*

$$\left(f^{-1}_{\uparrow,[a,+\infty[}\right)^{-1}_{\downarrow,[f(a),+\infty[} = f|^{\wedge}_{[a',+\infty[}.$$

Proof. We distinguish cases. First, let $t < a'$. We have that $f^{-1}_{\uparrow,[a,+\infty[}(y) = -\infty$ for $y < f(a)$, and $f^{-1}_{\uparrow,[a,+\infty[}(f(a)) = a'$. Therefore, for $t < a'$,

$$\left(f^{-1}_{\uparrow,[a,+\infty[}\right)^{-1}_{\downarrow,[f(a),+\infty[}(t) \overset{(14.5)}{=} +\infty,$$

where we used that $t < a' = f^{-1}_{\uparrow,[a,+\infty[}(f(a))$. Note that this is true even for all $t \in]a, a'[$. On the other hand,

$$f|^{\wedge}_{[a',+\infty[}(t) = +\infty.$$

Now, in the following, let $t \geq a'$. We can follow along the steps of Lemma 4.12, replacing 0 with a'. Inserting the definitions of lower and upper pseudoinverses yields

$$\begin{aligned}\left(f^{-1}_{\uparrow,[a,+\infty[}\right)^{-1}_{\downarrow,[f(a),+\infty[}(t) &= \inf\left\{x \geq f(a) \mid f^{-1}_{\uparrow,[a,+\infty[}(x) \geq t\right\}\\ &= \inf\{x \geq f(a) \mid \sup\{s \geq a \mid f(s) \leq x\} \geq t\}\\ &= \inf\left\{x \geq f(a) \mid \sup\left\{s \geq a' \mid f(s) \leq x\right\} \geq t\right\},\end{aligned}$$

where we used in the second line that $t \geq a' \geq a$. We introduce notation for the two sets appearing in the equation and define

$$\begin{aligned}M_t &:= \left\{x \geq f(a) \mid \sup\left\{s \geq a' \mid f(s) \leq x\right\} \geq t\right\},\\ M_{t,x} &:= \left\{s \geq a' \mid f(s) \leq x\right\}.\end{aligned}$$

We prove the claim by showing that $f(t) = \inf\{M_t\}$. Then, we show that $f(t) \in M_t$, and that if $\hat{x} < f(t)$, then $\hat{x} \notin M_t$. This would finish the proof as $f^{-1}_{\uparrow,[a,+\infty[}$ is right-continuous and hence an infimum of the set $M_t = \left\{x \geq f(a) \mid f^{-1}_{\uparrow,[a,+\infty[}(x) \geq t\right\}$ must also be an element of M_t.

If $x = f(t)$, the condition to be an element of M_t becomes

$$\sup\{M_{t,x}\} = \sup\left\{M_{t,f(t)}\right\} = \sup\{s \geq a' \mid f(s) \leq f(t)\} \geq t.$$

This supremum is either equal to t or equal to $s^* > t$ such that $f(s^*) = f(t)$. In both cases, the condition of the supremum to be greater than or equal to t is therefore satisfied, and hence we conclude that $f(t) \in M_t$.

For an $\hat{x} < f(t)$, the condition is equal to

$$\sup\{M_{t,\hat{x}}\} = \sup\{s \geq a' \mid f(s) \leq \hat{x}\} \geq t.$$

Let $s^* := \sup\{M_{t,\hat{x}}\}$ be the supremum. Since f is left-continuous, we have that $s^* \in M_{t,\hat{x}}$, and, therefore $f(s^*) \leq \hat{x}$. Combining both assumptions yields

$$f(s^*) \leq \hat{x} < f(t).$$

Since f is non-decreasing, this implies that $s^* < t$. Hence, with $\hat{x}$, the condition for membership in M_t cannot be satisfied, and we have that $\hat{x} \notin M_t$. □

Proposition 14.20. *Let $f \in \mathcal{U}$ be left-continuous and non-decreasing on the interval $I = [a, +\infty[$. Let $a' := \sup\{t \geq a \mid f(t) = f(a)\} \geq a$. Then*

$$\left[\left(f^{-1}_{\uparrow,[a,+\infty[}\right)^{-1}_{\downarrow,[f(a),+\infty[}\right]_a = f|^{\wedge}_{[a,+\infty[}. \tag{14.20}$$

Proof. Note that $a' \geq a$. We want to show that

$$\left[\left(f^{-1}_{\uparrow,[a,+\infty[}\right)^{-1}_{\downarrow,[f(a),+\infty[}\right]_a (t) = f|^{\wedge}_{[a,+\infty[}(t)$$

for all $t \in \mathbb{Q}_+$.

For $t \geq a'$, we have that

$$\left[\left(f^{-1}_{\uparrow,[a,+\infty[}\right)^{-1}_{\downarrow,[f(a),+\infty[}\right]_a (t) \overset{(14.19)}{=} \left(f^{-1}_{\uparrow,[a,+\infty[}\right)^{-1}_{\downarrow,[f(a),+\infty[}(t).$$

Together with $t \geq a' \geq a$, by Proposition 14.19 it follows that

$$\left[\left(f^{-1}_{\uparrow,[a,+\infty[}\right)^{-1}_{\downarrow,[f(a),+\infty[}\right]_a (t) = f(t) = f|^{\wedge}_{[a,+\infty[}(t).$$

Next, we consider $t < a$. Again, we have that

$$\left[\left(f^{-1}_{\uparrow,[a,+\infty[}\right)^{-1}_{\downarrow,[f(a),+\infty[}\right]_a (t) \overset{(14.19)}{=} \left(f^{-1}_{\uparrow,[a,+\infty[}\right)^{-1}_{\downarrow,[f(a),+\infty[}(t)$$

and thus, we obtain

$$\left[\left(f^{-1}_{\uparrow,[a,+\infty[}\right)^{-1}_{\downarrow,[f(a),+\infty[}\right]_a(t) = +\infty = f|^{\wedge}_{[a,+\infty[}(t).$$

The last remaining case is $t \in [a, a'[$. As $a' \geq a$, we have that

$$\begin{aligned}\left[\left(f^{-1}_{\uparrow,[a,+\infty[}\right)^{-1}_{\downarrow,[f(a),+\infty[}\right]_a(t) &\overset{(14.19)}{=} \left(f^{-1}_{\uparrow,[a,+\infty[}\right)^{-1}_{\downarrow,[f(a),+\infty[}(a') \\ &\overset{(14.5)}{=} \inf\left\{y \geq f(a) \mid f^{-1}_{\uparrow,[a,+\infty[}(y) \geq a'\right\} \\ &= f(a),\end{aligned}$$

where we used in the last line that $f^{-1}_{\uparrow,[a,+\infty[}(f(a)) = a'$. On the other hand, we have

$$f|^{\wedge}_{[a,+\infty[}(t) = f(t).$$

If $a' = a$, $[a, a'[$ is just a single point and the claim follows immediately. If $a' > a$, the functions f is constant on $[a, a'[$ and thus, $f(t) = f(a)$. Combining all cases concludes the proof. □

F.2.2 Isomorphism of restricted (max,+) convolution

Lemma 14.23. *Let $f \in \mathcal{U}$ be non-decreasing and $I = [a, +\infty[\subset \mathbb{Q}_+$. Let $x \in I$ and $y \geq f(a)$. If $f(x) \geq y$, then $f^{-1}_{\downarrow,I}(y) \leq x$.*

Proof. The proof follows mostly along the lines of [Lie17, p. 62].

Since $x \in I$ and $y \geq f(a)$, applying $f^{-1}_{\downarrow,I}$ to $f(x) \geq y$ preserves the inequality (we are in the infimum case of Equation (14.5)) and thus it follows that

$$f^{-1}_{\downarrow,I}(f(x)) \geq f^{-1}_{\downarrow,I}(y).$$

Moreover, since $x \in I$, it holds that

$$\begin{aligned}f^{-1}_{\downarrow,I}(f(x)) &= \inf\{z \in I \mid f(z) \geq f(x)\} \\ &\leq x.\end{aligned}$$

This finishes the proof. □

Proposition 14.24. *Let f and g be non-decreasing and right-continuous functions of $\mathcal{U}$. Then, for any $t \in \left[T_f + T_g, +\infty\right[$ it exists $s^* \in [T_f, t - T_g]$ such that*

$$\begin{aligned}(f^{\vee}_p \overline{\otimes} g^{\vee}_p)(t) &= \sup_{0 \leq s \leq t}\left\{f^{\vee}_p(s) + g^{\vee}_p(t-s)\right\} \\ &= \sup_{T_f \leq s \leq t - T_g}\left\{f^{\vee}_p(s) + g^{\vee}_p(t-s)\right\} \\ &= f(s^*) + g(t - s^*)\end{aligned}$$

In other words, the supremum is attainable.

Proof. The proof follows along the lines of [BBL18, pp. 48], adapted to functions restricted to the pseudo-periodic parts.

Fix $t \geq 0$ and define $F_t(s) := f_p^{\vee}(s) + g_p^{\vee}(t-s)$. Let $(s_n)_{n \in \mathbb{N}}$ be a sequence such that $F_t(s_n)$ is increasing and converges to

$$\sup_{0 \leq s \leq t} \left\{ f_p^{\vee}(s) + g_p^{\vee}(t-s) \right\} = \sup_{T_f \leq s \leq t - T_g} \{ f(s) + g(t-s) \}.$$

Similar to the (min,+) convolution [BT08, Proposition 7], one can show that the limit, the (max,+) convolution $\sup_{T_f \leq s \leq t-T_g} \{f(s) + g(t-s)\}$, is in $\mathbb{Q}$. As for all n, it holds that $s_n \in \left[T_f, t - T_g\right]$. In other words, we have an infinite sequence on a closed and bounded interval. Therefore, there exists a subsequence u_n that converges to $\bar{u} \in \left[T_f, t - T_g\right] \subset \mathbb{R}_+$ (Bolzano-Weierstrass Theorem for real sequences). Moreover, by the Lemma C.1, this limit $\bar{u} \in \mathbb{Q}_+$. By the Monotone Subsequence Theorem, there exists a monotonic subsequence $(v_n)_{n \in \mathbb{N}}$ (of this subsequence u_n) with the same limit $\bar{u}$. Without loss of generality, this sequence is decreasing (if this sequence is increasing, we can exchange the role of f and g and replace v_n with $t - v_n$). As f is right-continuous, $\lim_{n \to \infty} f(v_n) = f(\bar{v})$, and, as g is non-decreasing, $\lim_{n \to \infty} g(t - v_n) \leq g(t - \bar{v})$. Therefore,

$$\lim_{n \to \infty} F_t(v_n) = \lim_{n \to \infty} f(v_n) + g(t - v_n) \leq f(\bar{u}) + g(t - \bar{u}) = F(\bar{u}).$$

Thus, F_t reaches its maximum on $[T_f, t - T_g] \subseteq [0, t]$. □

Theorem 14.25. *Let f and g be right-continuous, non-decreasing UPP functions. Then*

$$\left(f_p^{\vee} \overline{\otimes} g_p^{\vee} \right)^{-1}_{\downarrow, \left[T_f + T_g, +\infty\right[} = \left(f_{\downarrow,p}^{-1} \otimes g_{\downarrow,p}^{-1} \right). \tag{14.23}$$

Proof. The proof follows mostly along the lines of Theorem 14.18, once adapted for (max,+) algebra.

First, we consider the case $x < f(T_f) + g(T_g)$. We calculate

$$\left(f_p^{\vee} \overline{\otimes} g_p^{\vee} \right) (T_f + T_g) = f(T_f) + g(T_g) > x,$$

where we used for the equality that $f_p^{\vee}(t) = -\infty$ for $t < T_f$ and $g_p^{\vee}(t) = -\infty$ for $t < T_g$. Therefore, by the definition of the lower pseudoinverse over an interval, we know for all $x < f(T_f) + g(T_g)$ that

$$\left(f_p^{\vee} \overline{\otimes} g_p^{\vee} \right)^{-1}_{\downarrow, \left[T_f + T_g, +\infty\right[}(x) \overset{(14.5)}{=} +\infty.$$

Similarly, we obtain

$$\left(f_{\downarrow,p}^{-1} \otimes g_{\downarrow,p}^{-1} \right)(y) = +\infty$$

for all $y < f(T_f) + g(T_g)$, since we know that $f^{-1}_{\downarrow,p}(y) = +\infty$ for $y < f(T_f)$ and $g^{-1}_{\uparrow,p}(y) = -\infty$ for $y < g(T_g)$, respectively.

Now, let $x \geq f(T_f) + g(T_g)$. First, we show that

$$\left(f_p^{\vee} \overline{\otimes} g_p^{\vee}\right)^{-1}_{\downarrow,\left[T_f+T_g,+\infty\right[} \geq \left(f^{-1}_{\downarrow,p} \otimes g^{-1}_{\downarrow,p}\right)(x).$$

Define

$$s_t^* := \operatorname*{arg\,sup}_{T_f \leq s \leq t-T_g} \{f_p^{\vee}(s) + g_p^{\vee}(t-s)\}$$

and $y_t^* := f_p^{\vee}(s_t^*)$. Note that, by Proposition 14.24, s_t^* exists and is in $[T_f, t-T_g]$ since $f_p^{\vee}$ and $g_p^{\vee}$ are right-continuous over $\left[T_f, +\infty\right[$ and $\left[T_g, +\infty\right[$, respectively. We continue with

$$\begin{aligned}
&\left(f_p^{\vee} \overline{\otimes} g_p^{\vee}\right)^{-1}_{\downarrow,\left[T_f+T_g,+\infty\right[} \\
&= \inf\left\{t \in \left[T_f+T_g,+\infty\right[\mid \left(f_p^{\vee} \overline{\otimes} g_p^{\vee}\right)(t) \geq x\right\} \\
&= \inf\left\{t \in \left[T_f+T_g,+\infty\right[\mid \sup_{0 \leq s \leq t}\left\{f_p^{\vee}(s) + g_p^{\vee}(t-s)\right\} \geq x\right\} \\
&= \inf\left\{t \in \left[T_f+T_g,+\infty\right[\mid \sup_{T_f \leq s \leq t-T_g}\left\{f_p^{\vee}(s) + g_p^{\vee}(t-s)\right\} \geq x\right\} \\
&= \inf\left\{t \in \left[T_f+T_g,+\infty\right[\mid \underbrace{f_p^{\vee}(s_t^*)}_{=y_t^*} + g_p^{\vee}(t-s_t^*) \geq x\right\} \\
&= \inf\left\{t \in \left[T_f+T_g,+\infty\right[\mid f_p^{\vee}(s_t^*) \geq y_t^* \text{ and } g_p^{\vee}(t-s_t^*) \geq x - y_t^*\right\} \\
&\overset{\text{(Lemma 14.23)}}{\geq} \inf\left\{t \in \left[T_f+T_g,+\infty\right[\mid f^{-1}_{\downarrow,p}(y_t^*) \leq s_t^* \text{ and } g^{-1}_{\downarrow,p}(x-y_t^*) \leq t - s_t^*\right\} \\
&\geq \inf\left\{t \in \left[T_f+T_g,+\infty\right[\mid f^{-1}_{\downarrow,p}(y_t^*) + g^{-1}_{\downarrow,p}(x-y_t^*) \leq t\right\} \\
&\geq \inf\left\{t \in \left[T_f+T_g,+\infty\right[\mid \inf_{0 \leq z \leq x}\left\{f^{-1}_{\downarrow,p}(z) + g^{-1}_{\downarrow,p}(x-z)\right\} \leq t\right\} \\
&= \inf\left\{t \in \left[T_f+T_g,+\infty\right[\mid \left(f^{-1}_{\downarrow,p} \otimes g^{-1}_{\downarrow,p}\right)(x) \leq t\right\} \\
&\geq \left(f^{-1}_{\downarrow,p} \otimes g^{-1}_{\downarrow,p}\right)(x)
\end{aligned}$$

where we used again that $\left(f^{-1}_{\downarrow,p} \otimes g^{-1}_{\downarrow,p}\right)(x) \geq T_f + T_g$ for all $x \geq f(T_f) + g(T_g)$. Moreover, we used that

$$\left(f^{-1}_{\downarrow,p}(y_t^*) \leq s_t^* \text{ and } g^{-1}_{\downarrow,p}(x-y_t^*) \leq t - s_t^*\right) \Rightarrow f^{-1}_{\downarrow,p}(y_t^*) + g^{-1}_{\downarrow,p}(x-y_t^*) \leq t$$

when replacing the "and" by the sum. For more details about this step, see Appendix G.3.

For the reverse direction, we derive

$$\begin{aligned}
&\left(f_{\downarrow,p}^{-1} \otimes g_{\downarrow,p}^{-1}\right)(x)\\
&= \inf_{0\le z\le x} \left\{ f_{\downarrow,p}^{-1}(z) + g_{\downarrow,p}^{-1}(z-x) \right\}\\
&= \inf_{f(T_f)\le z\le x-g(T_g)} \left\{ f_{\downarrow,p}^{-1}(z) + g_{\downarrow,p}^{-1}(z-x) \right\}\\
&= \inf_{f(T_f)\le z\le x-g(T_g)} \left\{ \inf\left\{\tau \in [T_f,+\infty[\mid f(\tau)\ge z\right\} + \inf\left\{s\in [T_g,+\infty[\mid g(s)\ge x-z\right\}\right\}\\
&= \inf_{f(T_f)\le z\le x-g(T_g)} \left\{ \inf\left\{\tau \in [T_f,+\infty[\mid f_p^{\vee}(\tau)\ge z\right\} + \inf\left\{s\in [T_g,+\infty[\mid g_p^{\vee}(s)\ge x-z\right\}\right\}\\
&= \inf_{f(T_f)\le z\le x-g(T_g)} \left\{ \inf\left\{t \in [T_f+T_g,+\infty[\mid t=\tau+s \text{ and } f_p^{\vee}(\tau)\ge z \text{ and } g_p^{\vee}(s)\ge x-z\right\}\right\}\\
&\ge \inf_{f(T_f)\le z\le x-g(T_g)} \left\{ \inf\left\{t \in [T_f+T_g,+\infty[\mid t=\tau+s \text{ and } f_p^{\vee}(\tau)+g_p^{\vee}(s)\ge x\right\}\right\}\\
&= \inf_{f(T_f)\le z\le x-g(T_g)} \left\{ \inf\left\{t \in [T_f+T_g,+\infty[\mid \sup_{0\le\tau\le t}\left\{f_p^{\vee}(\tau)+g_p^{\vee}(t-\tau)\right\}\ge x\right\}\right\}\\
&= \inf_{f(T_f)\le z\le x-g(T_g)} \left\{ \inf\left\{t \in [T_f+T_g,+\infty[\mid \left(f_p^{\vee}\,\overline{\otimes}\,g_p^{\vee}\right)(t)\ge x\right\}\right\}\\
&= \inf\left\{t \in [T_f+T_g,+\infty[\mid \left(f_p^{\vee}\,\overline{\otimes}\,g_p^{\vee}\right)(t)\ge x\right\}\\
&= \left(f_p^{\vee}\,\overline{\otimes}\,g_p^{\vee}\right)^{-1}_{\downarrow,\left[T_f+T_g,+\infty\right[}(x),
\end{aligned}$$

where, again, we used that $\left(f_p^{\vee}\,\overline{\otimes}\,g_p^{\vee}\right)^{-1}_{\downarrow,\left[T_f+T_g,+\infty\right[}(x) \ge T_f+T_g$ for $x\ge f(T_f)+g(T_g)$. This finishes the proof. □

Proposition 14.26. *Let $f\in\mathcal{U}$ be right-continuous and non-decreasing on the interval $I=[a,+\infty[$. Then*

$$\left(f^{-1}_{\downarrow,[a,+\infty[}\right)^{-1}_{\uparrow,[f(a),+\infty[} = f|^{\vee}_{[a,+\infty[}. \tag{14.24}$$

Proof. We distinguish cases. First, let $t<a$. We have that $f^{-1}_{\downarrow,[a,+\infty[}(y)=+\infty$ for $y<f(a)$, and $f^{-1}_{\downarrow,[a,+\infty[}(f(a))=a$. Therefore, for $t<a$,

$$\left(f^{-1}_{\downarrow,[a,+\infty[}\right)^{-1}_{\uparrow,[f(a),+\infty[}(t) \overset{(14.6)}{=} -\infty,$$

where we used that $t<a=f^{-1}_{\downarrow,[a,+\infty[}(f(a))$. On the other hand,

$$f|^{\vee}_{[a,+\infty[}(t) = -\infty.$$

Now, in the following, let $t \geq a$. We can follow along the lines of Lemma 4.13, replace 0 with a. Inserting the definitions of lower and upper pseudoinverses yields

$$\begin{aligned}\left(f^{-1}_{\downarrow,[a,+\infty[}\right)^{-1}_{\uparrow,[f(a),+\infty[}(t) &= \sup\left\{x \geq f(a) \mid f^{-1}_{\downarrow,[a,+\infty[}(x) \leq t\right\} \\ &= \sup\left\{x \geq f(a) \mid \inf\left\{s \geq a \mid f(s) \geq x\right\} \leq t\right\}.\end{aligned}$$

We introduce notation for the two sets appearing in the equation and define

$$\begin{aligned}M_t &:= \left\{x \geq f(a) \mid \inf\left\{s \geq a' \mid f(s) \geq x\right\} \leq t\right\}, \\ M_{t,x} &:= \left\{s \geq a' \mid f(s) \geq x\right\}.\end{aligned}$$

We prove the claim by showing that $f(t) = \sup\left\{M_t\right\}$. Then, we show that $f(t) \in M_t$, and that if $\hat{x} > f(t)$, then $\hat{x} \notin M_t$. This would finish the proof as $f^{-1}_{\downarrow,[a,+\infty[}$ is left-continuous and hence a supremum of the set $M_t = \left\{x \geq f(a) \mid f^{-1}_{\downarrow,[a,+\infty[}(x) \leq t\right\}$ must also be an element of M_t.

If $x = f(t)$, the condition to be an element of M_t becomes

$$\inf\left\{M_{t,x}\right\} = \inf\left\{M_{t,f(t)}\right\} = \inf\left\{s \geq a \mid f(s) \geq f(t)\right\} \leq t.$$

This infimum is either equal to t or equal to $s^* < t$ such that $f(s^*) = f(t)$. In both cases, the condition of the infimum to be less than or equal to t is therefore satisfied, and hence we conclude that $f(t) \in M_t$.

For an $\hat{x} > f(t)$, the condition is equal to

$$\inf\left\{M_{t,\hat{x}}\right\} = \inf\left\{s \geq a \mid f(s) \geq \hat{x}\right\} \leq t.$$

Let $s^* := \inf\left\{M_{t,\hat{x}}\right\}$ be the infimum. Since f is right-continuous, we have that $s^* \in M_{t,\hat{x}}$, and, therefore $f(s^*) \geq \hat{x}$. Combining both assumptions yields

$$f(s^*) \geq \hat{x} > f(t).$$

Since f is non-decreasing, this implies that $s^* > t$. Hence, with $\hat{x}$, the condition for membership in M_t cannot be satisfied, and we have that $\hat{x} \notin M_t$. □

F.3 Exploiting the isomorphism to speed up the (min,+) convolution

Theorem 14.29. *Let $f, g \in \mathcal{U}$ be left-continuous and non-decreasing functions. Let*

$$k_{c_f} := \frac{\operatorname{lcm}(c_f, c_g)}{c_f}, \tag{14.27}$$

$$k_{c_g} := \frac{\operatorname{lcm}(c_f, c_g)}{c_g}. \tag{14.28}$$

Then, $f_p^\wedge \otimes g_p^\wedge$ is again $\in \mathcal{U}$ with

$$T_{\otimes_{pp}} = \sup\left\{t \geq T_f + T_g \mid f_p^\wedge \otimes g_p^\wedge(t) \leq f(T_f) + g(T_g) + \operatorname{lcm}\left(c_f, c_g\right)\right\}, \tag{14.29}$$

$$d_{\otimes_{pp}} = \max\left(\frac{d_f}{c_f}, \frac{d_g}{c_g}\right) \cdot \operatorname{lcm}\left(c_f, c_g\right) = \max\left(k_{c_g} \cdot d_g, k_{c_f} \cdot d_f\right), \tag{14.30}$$

$$c_{\otimes_{pp}} = \operatorname{lcm}\left(c_f, c_g\right). \tag{14.31}$$

Proof. Using Proposition 14.20, we have that

$$\begin{aligned}
f_p^\wedge \otimes g_p^\wedge &\overset{(14.20)}{=} \left[\left(\left(f_p^\wedge \otimes g_p^\wedge\right)^{-1}_{\uparrow,\left[T_f+T_g,+\infty\right[}\right)^{-1}_{\downarrow,\left[f(T_f)+g(T_g),+\infty\right[}\right]_{(T_f+T_g)} \\
&\overset{(14.18)}{=} \left[\left(f^{-1}_{\uparrow,p} \,\overline{\otimes}\, g^{-1}_{\uparrow,p}\right)^{-1}_{\downarrow,\left[f(T_f)+g(T_g),+\infty\right[}\right]_{(T_f+T_g)},
\end{aligned}$$

where we used Theorem 14.18 in the second line. For the inner part (the (max,+) convolution), we obtain for all $x \geq f(T_f) + g(T_g) + c''_{\otimes_{pp}} = f(T_f) + g(T_g) + \operatorname{lcm}\left(c_f, c_g\right)$ (Equation (14.4)) that

$$\begin{aligned}
&\left(f^{-1}_{\uparrow,p} \,\overline{\otimes}\, g^{-1}_{\uparrow,p}\right)\left(x + c''_{\otimes_{pp}}\right) \\
&= \sup_{0 \leq u \leq x + c''_{\otimes_{pp}}} \left\{f^{-1}_{\uparrow,p}(u) + g^{-1}_{\uparrow,p}(x + c''_{\otimes_{pp}} - u)\right\} \\
&= \sup_{f(T_f) \leq u \leq x + c''_{\otimes_{pp}} - g(T_g)} \left\{f^{-1}_{\uparrow,p}(u) + g^{-1}_{\uparrow,p}(x + c''_{\otimes_{pp}} - u)\right\} \\
&= \sup_{f(T_f) \leq u \leq x - g(T_g)} \left\{f^{-1}_{\uparrow,p}(u) + g^{-1}_{\uparrow,p}(x + c''_{\otimes_{pp}} - u)\right\} \\
&\quad \vee \sup_{x - g(T_g) \leq u \leq x + c''_{\otimes_{pp}} - g(T_g)} \left\{f^{-1}_{\uparrow,p}(u) + g^{-1}_{\uparrow,p}(x + c''_{\otimes_{pp}} - u)\right\} \\
&\overset{\left(x - g(T_g) \geq f(T_f) + c''_{\otimes_{pp}}\right)}{=} \sup_{f(T_f) \leq u \leq x - g(T_g)} \left\{f^{-1}_{\uparrow,p}(u) + g^{-1}_{\uparrow,p}(x + c''_{\otimes_{pp}} - u)\right\} \\
&\quad \vee \sup_{f(T_f) + c''_{\otimes_{pp}} \leq u \leq x + c''_{\otimes_{pp}} - g(T_g)} \left\{f^{-1}_{\uparrow,p}(u) + g^{-1}_{\uparrow,p}(x + c''_{\otimes_{pp}} - u)\right\},
\end{aligned}$$

where we used the definition of the (max,+) convolution in the second line and the fact that we only consider the periodic phase in the third line. In the last two lines, we exploited that the supremum does not change since the intervals now overlap.

We continue by substituting $v := x + c''_{\otimes_{pp}} - u$:

$$\begin{aligned}
&\left(f^{-1}_{\uparrow,p} \overline{\otimes} g^{-1}_{\uparrow,p}\right)\left(x + c''_{\otimes_{pp}}\right) \\
\overset{(v:=x+c''_{\otimes_{pp}}-u)}{=}\; & \sup_{f(T_f)\le u \le x-g(T_g)} \left\{ f^{-1}_{\uparrow,p}(u) + g^{-1}_{\uparrow,p}(x + \underbrace{\mathrm{lcm}(c_f, c_g)}_{=k_{c_g}c_g} - u) \right\} \\
& \vee \sup_{g(T_g)\le v \le x-f(T_f)} \left\{ f^{-1}_{\uparrow,p}(x + \underbrace{\mathrm{lcm}(c_f, c_g)}_{=k_{c_f}c_f} - v) + g^{-1}_{\uparrow,p}(v) \right\} \\
\overset{\left(x-u\ge g(T_g), x-v\ge f(T_f)\right)}{=}\; & \sup_{f(T_f)\le u \le x-g(T_g)} \left\{ f^{-1}_{\uparrow,p}(u) + g^{-1}_{\uparrow,p}(x-u) \right\} + k_{c_g} d_g \\
& \vee \sup_{g(T_g)\le v \le x-f(T_f)} \left\{ f^{-1}_{\uparrow,p}(x-v) + g^{-1}_{\uparrow,p}(v) \right\} + k_{c_f} d_f \\
=\; & \left(f^{-1}_{\uparrow,p} \overline{\otimes} g^{-1}_{\uparrow,p}\right)(x) + \max\left(k_{c_g} \cdot d_g, k_{c_f} \cdot d_f\right) \\
=\; & \left(f^{-1}_{\uparrow,p} \overline{\otimes} g^{-1}_{\uparrow,p}\right)(x) + \max\left(\frac{d_f}{c_f}, \frac{d_g}{c_g}\right) \cdot \mathrm{lcm}(c_f, c_g),
\end{aligned}$$

where we used Theorem 10.4 in the fourth line. It follows then that

$$T_{\overline{\otimes}_p^{-1}} = f(T_f) + g(T_g) + \mathrm{lcm}(c_f, c_g), \tag{F.3}$$

$$d_{\overline{\otimes}_p^{-1}} = c''_{\otimes_{pp}} = \mathrm{lcm}(c_f, c_g), \tag{F.4}$$

$$c_{\overline{\otimes}_p^{-1}} = \max\left(k_{c_g} \cdot d_g, k_{c_f} \cdot d_f\right). \tag{F.5}$$

Next, for the outer part we consider the lower pseudoinverse of the above result, restricted to the interval $\left[f(T_f) + g(T_g), +\infty\right[$. From Lemmas 14.11 and 14.13, it follows that

$$\begin{aligned}
T''_{\otimes_{pp}} &\overset{(14.17)}{=} \left(f_p^{\wedge} \otimes g_p^{\wedge}\right)^{-1}_{\uparrow,\left[T_f+T_g,+\infty\right[}\left(T_{\overline{\otimes}_p^{-1}}\right) \\
&\overset{(14.6)}{=} \sup\left\{ t \ge T_f + T_g \mid f_p^{\wedge} \otimes g_p^{\wedge}(t) \le T_{\overline{\otimes}_p^{-1}} \right\}.
\end{aligned}$$

From Theorem 14.9 it follows also that

$$\begin{aligned}
d''_{\otimes_{pp}} &= c_{\overline{\otimes}_p^{-1}} = \max\left(k_{c_g} \cdot d_g, k_{c_f} \cdot d_f\right), \\
c''_{\otimes_{pp}} &= d_{\overline{\otimes}_p^{-1}} = \mathrm{lcm}(c_f, c_g).
\end{aligned}$$

This finishes the proof. □

Corollary 14.30. *Let f and $g \in \mathcal{U}$ which are neither UC nor UI, and are left-continuous and non-decreasing in $\left[T_f, +\infty\right[$ and $\left[T_g, +\infty\right[$, respectively. Let $k_{c_f} := \frac{\operatorname{lcm}\left(c_f, c_g\right)}{c_f}$, $k_{c_g} := \frac{\operatorname{lcm}\left(c_f, c_g\right)}{c_g}$. Then, to compute $f_p^\wedge \otimes g_p^\wedge$ via the (max,+) isomorphism (Equation (14.22)), it is sufficient to use $S_{f_p^\wedge}^{I_{f_p^\wedge}}$ and $S_{g_p^\wedge}^{I_{g_p^\wedge}}$ with*

$$\begin{aligned} I_{f_p^\wedge} &= \left[T_f, T_f' + 2 \cdot k_{c_f} \cdot d_f\right], \\ I_{g_p^\wedge} &= \left[T_g, T_g' + 2 \cdot k_{c_g} \cdot d_g\right]. \end{aligned} \tag{14.34}$$

where we used[2]

$$\begin{aligned} T_f' &= \sup\left\{t \geq T_f \mid f(t) = f(T_f)\right\}, \\ T_g' &= \sup\left\{t \geq T_g \mid g(t) = g(T_g)\right\}. \end{aligned}$$

Proof. The proof is based on using Proposition 14.20, as we did in the proof of Corollary 14.30. We thus compute $f_p^\wedge \otimes g_p^\wedge$ through $f_{\uparrow,p}^{-1} \overline{\otimes} g_{\uparrow,p}^{-1}$.

In the proof of Corollary 14.30, we have shown that the latter has the following UPP properties:

$$\begin{aligned} T_{\overline{\otimes}_p^{-1}} &\overset{(F.3)}{=} f(T_f) + g(T_g) + \operatorname{lcm}\left(c_f, c_g\right), \\ d_{\overline{\otimes}_p^{-1}} &\overset{(F.4)}{=} \operatorname{lcm}\left(c_f, c_g\right), \\ c_{\overline{\otimes}_p^{-1}} &\overset{(F.5)}{=} \max\left(k_{c_g} \cdot d_g, k_{c_f} \cdot d_f\right). \end{aligned}$$

Thus, it is sufficient to compute $S_{\overline{\otimes}_p^{-1}}^{I_{\overline{\otimes}_p^{-1}}} = S_q^{I_q} \overline{\otimes} S_r^{I_r}$, where $q := f_{\uparrow,p}^{-1}$ and $r := g_{\uparrow,p}^{-1}$ with

$$I_{\overline{\otimes}_p^{-1}} = \left[f(T_f) + g(T_g), f(T_f) + g(T_g) + 2 \cdot \operatorname{lcm}\left(c_f, c_g\right)\right[, \tag{F.6}$$

$$I_{f_{\uparrow,p}^{-1}} = \left[f(T_f), f(T_f) + 2 \cdot \operatorname{lcm}\left(c_f, c_g\right)\right] = \left[f(T_f), f(T_f) + 2 \cdot k_{c_f} \cdot c_f\right], \tag{F.7}$$

$$I_{g_{\uparrow,p}^{-1}} = \left[g(T_g), g(T_g) + 2 \cdot \operatorname{lcm}\left(c_f, c_g\right)\right] = \left[g(T_g), g(T_g) + 2 \cdot k_{c_g} \cdot c_g\right]. \tag{F.8}$$

Next, we derive which values of $f_p^\wedge$ and $g_p^\wedge$ are sufficient in order to compute the values of $f_{\uparrow,p}^{-1}$ in $I_{f_{\uparrow,p}^{-1}}$ and $g_{\uparrow,p}^{-1}$ in $I_{g_{\uparrow,p}^{-1}}$. We focus, without loss of generality, on $f_p^\wedge$, and obtain via Lemma 14.15 that

$$\begin{aligned} f_{\uparrow,p}^{-1}(f(T_f)) &= \sup\left\{t \geq T_f \mid f(t) \leq f(T_f)\right\} \\ &= T_f' \end{aligned}$$

[2]The suprema are attainable since the functions are left-continuous over the respective intervals.

and using Theorem 10.4

$$\begin{aligned} f_{\uparrow,p}^{-1}(f(T_f) + 2 \cdot \operatorname{lcm}(c_f, c_g)) &= f_{\uparrow,p}^{-1}(f(T_f)) + 2 \cdot k_{c_f} \cdot d_f \\ &= T'_f + 2 \cdot k_{c_f} \cdot d_f. \end{aligned}$$

Hence, it is sufficient to use

$$\begin{aligned} &\left[T'_f, T'_f + 2 \cdot k_{c_f} \cdot d_f\right] \quad \text{for } f_{\uparrow,p}^{-1}, \\ &\left[T'_g, T'_g + 2 \cdot k_{c_g} \cdot d_g\right] \quad \text{for } g_{\uparrow,p}^{-1}. \end{aligned}$$

Thus, the above intervals enable us to compute $f_{\uparrow,p}^{-1} \overline{\otimes} g_{\uparrow,p}^{-1}$. Then, to compute the lower pseudoinverse, we do not require any additional value from f and g. We do so however for the last step, due to the loss of information implied by having T'_f and T'_g as left boundaries. Using Proposition 14.20, the reconstruction operator requires us to know that $f(t) = f(T_f)\ \forall T_f \leq t \leq T'_f$, we obtain

$$\begin{aligned} I'_{f_p^\wedge} &= \left[T_f, T'_f + 2 \cdot k_{c_f} \cdot d_f\right], \\ I'_{g_p^\wedge} &= \left[T_g, T'_g + 2 \cdot k_{c_g} \cdot d_g\right]. \end{aligned}$$

□

Theorem 14.31 (Mix and Match ((min,+) Convolution)). *Let f and $g \in \mathcal{U}$ which are neither UC nor UI, and are left-continuous and non-decreasing in $[T_f, +\infty[$ and $[T_g, +\infty[$, respectively. Let $I_{f_p^\wedge}, I_{g_p^\wedge}$ be the intervals sufficient to compute $f_p^\wedge \otimes g_p^\wedge$ according to Proposition 3.16, and let $I'_{f_p^\wedge}, I'_{g_p^\wedge}$ be the intervals sufficient to compute $f_{\uparrow,p}^{-1} \overline{\otimes} g_{\uparrow,p}^{-1}$ according to Corollary 14.30.*

Then $I_{f_p^\wedge} \cap I'_{f_p^\wedge}, I_{g_p^\wedge} \cap I'_{g_p^\wedge}$ are intervals sufficient to compute $f_p^\wedge \otimes g_p^\wedge$.

Proof. We distinguish four cases, based on the result of the intersections. In the first case, $I_{f_p^\wedge} \subseteq I'_{f_p^\wedge}$ and $I_{g_p^\wedge} \subseteq I'_{g_p^\wedge}$, in the second case $I_{f_p^\wedge} \supset I'_{f_p^\wedge}$ and $I_{g_p^\wedge} \subseteq I'_{g_p^\wedge}$, in the third $I_{f_p^\wedge} \subseteq I'_{f_p^\wedge}$ and $I_{g_p^\wedge} \supset I'_{g_p^\wedge}$, and finally in the fourth $I_{f_p^\wedge} \supset I'_{f_p^\wedge}$ and $I_{g_p^\wedge} \supset I'_{g_p^\wedge}$.

The first case is trivial, since the resulting intervals are sufficient due to Proposition 3.16. We now prove the result for the fourth case, while the second and third can be derived following the same steps.

Let $I_{f_p^\wedge} = [T_f, b_f]$, $I'_{f_p^\wedge} = [T_f, a_f]$ with $a_f < b_f$, and $I_{g_p^\wedge} = [T_f, b_g]$, $I'_{g_p^\wedge} = [T_f, a_g]$ with $a_g < b_g$. Moreover, we define $d_{\otimes_{pp}}$ according to (14.32), and $T_{\otimes_{pp}}$ according to (14.29).

We show now that the values of f in $]a_f, b_f]$ and g in $]a_g, b_g]$ are not necessary for the computation. Therefore, assume that this is not the case, i.e., there exists some

$t^* \in \left[T_f + T_g, T_{\otimes_{pp}} + d_{\otimes_{pp}}\right]$ such that

$$\inf_{0 \le s \le t^*, s \notin \left]a_f, b_f\right]} \left\{ f_p^{\wedge}(s) + g_p^{\wedge}(t^* - s) \right\} > f_p^{\wedge} \otimes g_p^{\wedge}(t^*)$$
$$= f_p^{\wedge}(s^*) + g_p^{\wedge}\left(t^* - s^*\right) =: z^*,$$

where either $s^* \in \left]a_f, b_f\right]$ or $t^* - s^* \in \left]a_g, b_g\right]$.

From $I'_{f_p^{\wedge}} = \left[T_f, a_f\right]$, $I'_{g_p^{\wedge}} = \left[T_g, a_g\right]$ and Lemma 14.15 it follows that

$$I_{f^{-1}_{\uparrow,p}} = \left[f(T_f), f(a_f)\right],$$
$$I_{g^{-1}_{\uparrow,p}} = \left[g(T_g), g(a_g)\right]$$

Let us consider now

$$\left(f_p^{\wedge} \otimes g_p^{\wedge}\right)^{-1}_{\uparrow}(z^*) = f^{-1}_{\uparrow,p} \overline{\otimes} g^{-1}_{\uparrow,p}(z^*).$$

We distinguish two cases: either $z^* \in I_{\overline{\otimes}_p^{-1}}$, computed according to Equation (F.6), or it is larger than the upper boundary of $I_{\overline{\otimes}_p^{-1}}$ (as it cannot be less than the lower boundary). In the first case, i.e., $z^* < f(T_f) + g(T_g) + 2 \cdot \mathrm{lcm}\left(c_f, c_g\right)$, we have

$$f^{-1}_{\uparrow,p} \overline{\otimes} g^{-1}_{\uparrow,p}(z^*) = \sup_{0 \le v \le z^*} \left\{ f^{-1}_{\uparrow,p}(v) + g^{-1}_{\uparrow,p}(z^* - v) \right\}$$
$$= f^{-1}_{\uparrow,p}(v^*) + g^{-1}_{\uparrow,p}\left(z^* - v^*\right),$$

where $v^* \in I'_{f^{-1}_{\uparrow,p}} = \left[f(T_f), f(a_f)\right]$, $z^* - v^* \in I'_{g^{-1}_{\uparrow,p}} = \left[g(T_g), g(a_g)\right]$ such that the supremum is attained. These elements exist since the upper pseudoinverses are right-continuous over their respective intervals (Lemma 14.8), thus their (max,+) convolution can always be attained (Proposition 14.24). Moreover, since $I'_{f^{-1}_{\uparrow,p}}$ and $I_{g^{-1}_{\uparrow,p}}$ are sufficient to compute $f^{-1}_{\uparrow,p} \overline{\otimes} g^{-1}_{\uparrow,p}(z)$ for any $z \in I_{\overline{\otimes}_p^{-1}}$ (Corollary 14.30), it follows that $v^* \in I'_{f^{-1}_{\uparrow,p}}$ and $z^* - v^* \in I_{g^{-1}_{\uparrow,p}}$, hence we do not need any value of f and g outside these intervals to perform the computation for z^* in particular. But, since the interval $\left]a_f, b_f\right]$ was not used to compute $I'_{f^{-1}_{\uparrow,p}}$, nor $\left]a_g, b_g\right]$ to compute $I'_{g^{-1}_{\uparrow,p}}$, this is a contradiction to the assumption that $s^* \in \left]a_f, b_f\right]$ or $t^* - s^* \in \left]a_g, b_g\right]$ are needed for the computation of $f_p^{\wedge} \otimes g_p^{\wedge}$.

In the second case ($z^* \ge f(T_f) + g(T_g) + 2 \cdot \mathrm{lcm}\left(c_f, c_g\right)$), $f^{-1}_{\uparrow,p} \overline{\otimes} g^{-1}_{\uparrow,p}(z^*)$ can be computed by applying the UPP property meaning that

$$f^{-1}_{\uparrow,p} \overline{\otimes} g^{-1}_{\uparrow,p}(z^*) = f^{-1}_{\uparrow,p} \overline{\otimes} g^{-1}_{\uparrow,p}\left(z^* - k \cdot d_{\overline{\otimes}_p^{-1}}\right) + k \cdot c_{\overline{\otimes}_p^{-1}}$$

for $d_{\overline{\otimes}_p^{-1}}$ and $c_{\overline{\otimes}_p^{-1}}$ described in Equation (F.4) and Equation (F.5), respectively, and some $k \in \mathbb{N}$ such that $z^* - k \cdot d_{\overline{\otimes}_p^{-1}} \in I_{\overline{\otimes}_p^{-1}}$. We can follow for the latter the same reasoning as in the first case, thus having the same contradiction. This concludes the proof. $\square$

F.4 Exploiting the isomorphism to speed up the (max,+) convolution

Proposition 14.2. *Let f and $g \in \mathcal{U}$ which are neither UC nor UI, and are right-continuous and non-decreasing. Then,*

$$d_{f \overline{\otimes} g} = \min\left(\frac{d_f}{c_f}, \frac{d_g}{c_g}\right) \cdot \operatorname{lcm}(c_f, c_g)\,, \tag{14.3}$$

$$c_{f \overline{\otimes} g} = \operatorname{lcm}(c_f, c_g) \tag{14.4}$$

are sufficient period length and height for $f \overline{\otimes} g$.

Proof. Since f and g are right-continuous, it holds by Equation (4.15) that

$$f \overline{\otimes} g = \left(f_{\downarrow}^{-1} \otimes g_{\downarrow}^{-1}\right)_{\uparrow}^{-1}.$$

Combining this with Proposition 3.17, we obtain for the inner function $f_{\downarrow}^{-1} \otimes g_{\downarrow}^{-1}$

$$d_{f_{\downarrow}^{-1} \otimes g_{\downarrow}^{-1}} = \operatorname{lcm}\left(d_{f_{\downarrow}^{-1}}, d_{g_{\downarrow}^{-1}}\right),$$

$$c_{f_{\downarrow}^{-1} \otimes g_{\downarrow}^{-1}} = \min\left(\frac{c_{f_{\downarrow}^{-1}}}{d_{f_{\downarrow}^{-1}}}, \frac{c_{g_{\downarrow}^{-1}}}{d_{g_{\downarrow}^{-1}}}\right) \cdot d_{f_{\downarrow}^{-1} \otimes g_{\downarrow}^{-1}}.$$

Using Theorem 10.2, we obtain for the period-length and height of the lower pseudoinverse

$$d_{f_{\downarrow}^{-1} \otimes g_{\downarrow}^{-1}} = \operatorname{lcm}(c_f, c_g)\,,$$

$$c_{f_{\downarrow}^{-1} \otimes g_{\downarrow}^{-1}} = \min\left(\frac{d_f}{c_f}, \frac{d_g}{c_g}\right) \cdot \operatorname{lcm}(c_f, c_g)\,.$$

Combining this with the outer function $(\cdot)_{\uparrow}^{-1}$, due to Theorem 10.4, we eventually obtain for $\left(f_{\downarrow}^{-1} \otimes g_{\downarrow}^{-1}\right)_{\uparrow}^{-1}$ that

$$\begin{aligned} d''_{f \overline{\otimes} g} &\overset{(4.15)}{=} d_{\left(f_{\downarrow}^{-1} \otimes g_{\downarrow}^{-1}\right)_{\uparrow}^{-1}} = c_{f_{\downarrow}^{-1} \otimes g_{\downarrow}^{-1}} = \min\left(\frac{d_f}{c_f}, \frac{d_g}{c_g}\right) \cdot \operatorname{lcm}(c_f, c_g), \\ c''_{f \overline{\otimes} g} &\overset{(4.15)}{=} c_{\left(f_{\downarrow}^{-1} \otimes g_{\downarrow}^{-1}\right)_{\uparrow}^{-1}} = d_{f_{\downarrow}^{-1} \otimes g_{\downarrow}^{-1}} = \operatorname{lcm}(c_f, c_g). \end{aligned}$$

This finishes the proof. □

Theorem 14.32. *Let f and $g \in \mathcal{U}$ which are neither UC nor UI, and are right-continuous and non-decreasing. Moreover, let f and g satisfy the assumptions of Lemma 14.11, such that $T_{f_{\downarrow,p}^{-1}} = f(T_f)$, $T_{g_{\downarrow,p}^{-1}} = g(T_g)$. Let*

$$k_{c_f} := \frac{\operatorname{lcm}(c_f, c_g)}{c_f}, \tag{14.35}$$

$$k_{c_g} := \frac{\operatorname{lcm}(c_f, c_g)}{c_g}. \tag{14.36}$$

Then, $f_p^{\vee} \overline{\otimes} g_p^{\vee}$ is again a function of $\mathcal{U}$ with

$$T_{\overline{\otimes}_{pp}} = \inf\left\{t \geq T_f + T_g \mid f_p^{\vee} \overline{\otimes} g_p^{\vee}(t) \geq f(T_f) + g(T_g) + \operatorname{lcm}(c_f, c_g)\right\}, \tag{14.37}$$

$$d_{\overline{\otimes}_{pp}} = \min\left(\frac{d_f}{c_f}, \frac{d_g}{c_g}\right) \cdot \operatorname{lcm}(c_f, c_g) = \min\left(k_{c_g} \cdot d_g, k_{c_f} \cdot d_f\right), \tag{14.38}$$

$$c_{\overline{\otimes}_{pp}} = \operatorname{lcm}(c_f, c_g). \tag{14.39}$$

Proof. Using Proposition 14.26, we have that

$$\begin{aligned} f_p^{\vee} \overline{\otimes} g_p^{\vee} &\overset{(14.24)}{=} \left(\left(f_p^{\vee} \overline{\otimes} g_p^{\vee}\right)_{\downarrow,\left[T_f+T_g,+\infty\right[}^{-1}\right)_{\uparrow,\left[f(T_f)+g(T_g),+\infty\right[}^{-1} \\ &\overset{(14.23)}{=} \left(f_{\downarrow,p}^{-1} \otimes g_{\downarrow,p}^{-1}\right)_{\uparrow,\left[f(T_f)+g(T_g),+\infty\right[}^{-1}, \end{aligned}$$

where we used Theorem 14.25 in the second line. For the inner part (the (min,+) convolution), we obtain for all $x \geq f(T_f) + g(T_g) + c''_{\overline{\otimes}_{pp}} = f(T_f) + g(T_g) + \operatorname{lcm}(c_f, c_g)$ (Equation (14.4)) that

$$
\begin{aligned}
&\left(f_{\downarrow,p}^{-1} \otimes g_{\downarrow,p}^{-1}\right)\left(x + c''_{\otimes pp}\right)\\
&= \inf_{0 \le u \le x + c''_{\otimes pp}} \left\{ f_{\downarrow,p}^{-1}(u) + g_{\downarrow,p}^{-1}(x + c''_{\otimes pp} - u) \right\}\\
&= \inf_{f(T_f) \le u \le x + c''_{\otimes pp} - g(T_g)} \left\{ f_{\downarrow,p}^{-1}(u) + g_{\downarrow,p}^{-1}(x + c''_{\otimes pp} - u) \right\}\\
&= \inf_{f(T_f) \le u \le x - g(T_g)} \left\{ f_{\downarrow,p}^{-1}(u) + g_{\downarrow,p}^{-1}(x + c''_{\otimes pp} - u) \right\}\\
&\quad \wedge \inf_{x - g(T_g) \le u \le x + c''_{\otimes pp} - g(T_g)} \left\{ f_{\downarrow,p}^{-1}(u) + g_{\downarrow}^{-1}(x + c''_{\otimes pp} - u) \right\}\\
&\overset{\left(x - g(T_g) \ge f(T_f) + c''_{\otimes pp}\right)}{=} \inf_{f(T_f) \le u \le x - g(T_g)} \left\{ f_{\downarrow,p}^{-1}(u) + g_{\downarrow,p}^{-1}(x + c''_{\otimes pp} - u) \right\}\\
&\quad \wedge \inf_{f(T_f) + c''_{\otimes pp} \le u \le x + c''_{\otimes pp} - g(T_g)} \left\{ f_{\downarrow,p}^{-1}(u) + g_{\downarrow,p}^{-1}(x + c''_{\otimes pp} - u) \right\},
\end{aligned}
$$

where we used the definition of the (min,+) convolution in the second line and the fact that we only consider the periodic phase in the third line. In the last two lines, we exploited that the infimum does not change since the intervals now overlap. We continue by substituting $v := x + c''_{\otimes pp} - u$:

$$
\begin{aligned}
&\left(f_{\downarrow,p}^{-1} \otimes g_{\downarrow,p}^{-1}\right)\left(x + c''_{\otimes pp}\right)\\
&\overset{(v := x + c''_{\otimes pp} - u)}{=} \inf_{f(T_f) \le u \le x - g(T_g)} \left\{ f_{\downarrow,p}^{-1}(u) + g_{\downarrow,p}^{-1}(x + \underbrace{\operatorname{lcm}(c_f, c_g)}_{= k_{c_g} c_g} - u) \right\}\\
&\quad \wedge \inf_{g(T_g) \le v \le x - f(T_f)} \left\{ f_{\downarrow,p}^{-1}(x + \underbrace{\operatorname{lcm}(c_f, c_g)}_{= k_{c_f} c_f} - v) + g_{\downarrow,p}^{-1}(v) \right\}\\
&\overset{\left(x - u \ge g(T_g),\, x - v \ge f(T_f)\right)}{=} \inf_{f(T_f) \le u \le x - g(T_g)} \left\{ f_{\downarrow,p}^{-1}(u) + g_{\downarrow,p}^{-1}(x - u) \right\} + k_{c_g} d_g\\
&\quad \wedge \inf_{g(T_g) \le v \le x - f(T_f)} \left\{ f_{\downarrow,p}^{-1}(x - v) + g_{\downarrow,p}^{-1}(v) \right\} + k_{c_f} d_f\\
&= \left(f_{\downarrow,p}^{-1} \otimes g_{\downarrow,p}^{-1}\right)(x) + \min\left(k_{c_g} \cdot d_g, k_{c_f} \cdot d_f\right)\\
&= \left(f_{\downarrow,p}^{-1} \otimes g_{\downarrow,p}^{-1}\right)(x) + \min\left(\frac{d_f}{c_f}, \frac{d_g}{c_g}\right) \cdot \operatorname{lcm}(c_f, c_g),
\end{aligned}
$$

where we used Theorem 10.2 and Lemma 14.11 in the fourth line. It follows then that

$$T_{\otimes_p^{-1}} = f(T_f) + g(T_g) + \operatorname{lcm}\left(c_f, c_g\right), \tag{F.9}$$

$$d_{\otimes_p^{-1}} = \operatorname{lcm}\left(c_f, c_g\right), \tag{F.10}$$

$$c_{\otimes_p^{-1}} = \min\left(k_{c_g} \cdot d_g, k_{c_f} \cdot d_f\right). \tag{F.11}$$

Next, for the outer part we consider the upper pseudoinverse of the above result, restricted to the interval $\left[f(T_f) + g(T_g), +\infty\right[$. From Theorem 14.10, it follows that

$$\begin{aligned} T''_{\overline{\otimes}_{pp}} &\overset{(14.12)}{=} \left(f_{\downarrow,p}^{-1} \otimes g_{\downarrow,p}^{-1}\right)\left(T_{\otimes_p^{-1}}\right) \\ &\overset{(14.23)}{=} \left(f_p^{\vee} \overline{\otimes} g_p^{\vee}\right)^{-1}_{\downarrow,\left[T_f+T_g,+\infty\right[}\left(T_{\otimes_p^{-1}}\right) \\ &\overset{(14.5)}{=} \inf\left\{x \geq T_f + T_g \mid f_p^{\vee} \overline{\otimes} g_p^{\vee}(t) \geq T_{\otimes_p^{-1}}\right\}. \end{aligned}$$

From Theorem 14.10 it follows also that

$$\begin{aligned} d''_{\overline{\otimes}_{pp}} &= c_{\otimes_p^{-1}} = \min\left(k_{c_g} \cdot d_g, k_{c_f} \cdot d_f\right), \\ c''_{\overline{\otimes}_{pp}} &= d_{\otimes_p^{-1}} = \operatorname{lcm}\left(c_f, c_g\right). \end{aligned}$$

This finishes the proof. □

Corollary 14.33. *Let f and $g \in \mathcal{U}$ which are neither UC nor UI, and are right-continuous and non-decreasing in $\left[T_f, +\infty\right[$ and $\left[T_g, +\infty\right[$, respectively. Moreover, let f and g satisfy the assumptions of Lemma 14.11, such that $T_{f_{\downarrow,p}^{-1}} = f(T_f)$, $T_{g_{\downarrow,p}^{-1}} = g(T_g)$.*

Let $k_{c_f} := \frac{\operatorname{lcm}\left(c_f, c_g\right)}{c_f}$, $k_{c_g} := \frac{\operatorname{lcm}\left(c_f, c_g\right)}{c_g}$. Then, to compute $f_p^{\vee} \overline{\otimes} g_p^{\vee}$ via the $(\min, +)$ isomorphism (Equation (14.26)), it is sufficient to use $S_{f_p^{\vee}}^{I'_{f_p^{\vee}}}$ and $S_{g_p^{\vee}}^{I'_{g_p^{\vee}}}$ with

$$\begin{aligned} I_{f_p^{\vee}} &= \left[T_f, T_f + 2 \cdot k_{c_f} \cdot d_f\right], \\ I_{g_p^{\vee}} &= \left[T_g, T_g + 2 \cdot k_{c_g} \cdot d_g\right]. \end{aligned} \tag{14.42}$$

Proof. The proof is based on using Proposition 14.26, as we did in the proof of Corollary 14.33. We thus compute $f_p^{\vee} \overline{\otimes} g_p^{\vee}$ through $f_{\downarrow,p}^{-1} \otimes g_{\downarrow,p}^{-1}$.

In the proof of Corollary 14.33, we have shown that the latter has the following UPP properties:

$$\begin{aligned} T_{\otimes_p^{-1}} &\overset{(F.9)}{=} f(T_f) + g(T_g) + \operatorname{lcm}\left(c_f, c_g\right), \\ d_{\otimes_p^{-1}} &\overset{(F.10)}{=} \operatorname{lcm}\left(c_f, c_g\right), \\ c_{\otimes_p^{-1}} &\overset{(F.11)}{=} \min\left(k_{c_g} \cdot d_g, k_{c_f} \cdot d_f\right). \end{aligned}$$

Thus, it is sufficient to compute $S_{\otimes_p^{-1}}^{I_{\otimes_p^{-1}}} = S_q^{I_q} \overline{\otimes} S_r^{I_r}$, where $q := f_{\downarrow,p}^{-1}$ and $r := g_{\downarrow,p}^{-1}$, and

$$I_{\otimes_p^{-1}} = \left[f(T_f) + g(T_g), f(T_f) + g(T_g) + 2 \cdot \operatorname{lcm}(c_f, c_g)\right[, \tag{F.12}$$

$$I_{f_{\downarrow,p}^{-1}} = \left[f(T_f), f(T_f) + 2 \cdot \operatorname{lcm}(c_f, c_g)\right] = \left[f(T_f), f(T_f) + 2 \cdot k_{c_f} \cdot c_f\right], \tag{F.13}$$

$$I_{g_{\downarrow,p}^{-1}} = \left[g(T_g), g(T_g) + 2 \cdot \operatorname{lcm}(c_f, c_g)\right] = \left[g(T_g), g(T_g) + 2 \cdot k_{c_g} \cdot c_g\right]. \tag{F.14}$$

Next, we derive which values of $f_p^{\vee}$ and $g_p^{\vee}$ in order to compute the values of $f_{\downarrow,p}^{-1}$ in $I_{f_{\downarrow,p}^{-1}}$ and $g_{\downarrow,p}^{-1}$ in $I_{g_{\downarrow,p}^{-1}}$. We focus, without loss of generality, on $f_p^{\vee}$, and obtain via Lemma 14.14 that

$$\begin{aligned} f_{\downarrow,p}^{-1}(f(T_f)) &= \inf\left\{t \geq T_f \mid f(t) \geq f(T_f)\right\} \\ &= T_f \end{aligned}$$

and, using Theorem 10.2 and Lemma 14.11,

$$\begin{aligned} f_{\downarrow,p}^{-1}\left(f(T_f) + 2 \cdot \operatorname{lcm}(c_f, c_g)\right) &= f_{\downarrow,p}^{-1}(f(T_f)) + 2 \cdot k_{c_f} \cdot d_f \\ &= T_f + 2 \cdot k_{c_f} \cdot d_f. \end{aligned}$$

Hence, it is sufficient to use

$$\begin{aligned} I'_{f_p^{\vee}} &= \left[T_f, T_f + 2 \cdot k_{c_f} \cdot d_f\right], \\ I'_{g_p^{\vee}} &= \left[T_g, T_g + 2 \cdot k_{c_g} \cdot d_g\right]. \end{aligned}$$

Thus, the above intervals enable us to compute $f_{\downarrow,p}^{-1} \otimes g_{\downarrow,p}^{-1}$. Then, to compute the upper pseudoinverse we do not require any additional value from f and g. □

Theorem 14.34 (Mix and Match ((max,+) Convolution))**.** *Let f and $g \in \mathcal{U}$ which are neither UC nor UI, and are right-continuous and non-decreasing in $\left[T_f, +\infty\right[$ and $\left[T_g, +\infty\right[$, respectively. Moreover, let f and g satisfy the assumptions of Lemma 14.11, such that $T_{f_{\downarrow,p}^{-1}} = f(T_f)$, $T_{g_{\downarrow,p}^{-1}} = g(T_g)$. Let $I_{f_p^{\vee}}, I_{g_p^{\vee}}$ be the intervals sufficient to compute $f_p^{\vee} \overline{\otimes} g_p^{\vee}$ according to Proposition 3.32, and let $I'_{f_p^{\vee}}, I'_{g_p^{\vee}}$ be the intervals sufficient to compute $f_{\downarrow,p}^{-1} \otimes g_{\downarrow,p}^{-1}$ according to Corollary 14.33.*

Then, $I_{f_p^{\vee}} \cap I'_{f_p^{\vee}}, I_{g_p^{\vee}} \cap I'_{g_p^{\vee}}$ are sufficient intervals to compute $f_p^{\vee} \overline{\otimes} g_p^{\vee}$.

Proof. We distinguish four cases, based on the result of the intersections. In the first case, $I_{f_p^{\vee}} \subseteq I'_{f_p^{\vee}}$ and $I_{g_p^{\vee}} \subseteq I'_{g_p^{\vee}}$, in the second case $I_{f_p^{\vee}} \supset I'_{f_p^{\vee}}$ and $I_{g_p^{\vee}} \subseteq I'_{g_p^{\vee}}$, in the third $I_{f_p^{\vee}} \subseteq I'_{f_p^{\vee}}$ and $I_{g_p^{\vee}} \supset I'_{g_p^{\vee}}$, and finally in the fourth $I_{f_p^{\vee}} \supset I'_{f_p^{\vee}}$ and $I_{g_p^{\vee}} \supset I'_{g_p^{\vee}}$.

The first case is trivial, since the resulting intervals are sufficient due to Proposition 3.32. We now prove the result for the fourth case, while the second and third can be derived following the same steps.

Let $I_{f_p^\vee} = [T_f, b_f]$, $I'_{f_p^\vee} = [T_f, a_f]$ with $a_f < b_f$, and $I_{g_p^\vee} = [T_f, b_g]$, $I'_{g_p^\vee} = [T_f, a_g]$ with $a_g < b_g$. Moreover, we define $d_{\overline{\otimes}_{pp}}$ according to Equation (14.40), and $T_{\overline{\otimes}_{pp}}$ according to (14.37).

We show now that the values of f in $]a_f, b_f]$ and g in $]a_g, b_g]$ are not necessary for the computation. Therefore, assume that this is not the case, i.e., there exists some $t^* \in \left[T_f + T_g, T_{\overline{\otimes}_{pp}} + d_{\overline{\otimes}_{pp}}\right]$ such that

$$\begin{aligned}\sup_{0 \le s \le t^*, s \notin]a_f, b_f]} \left\{ f_p^\vee(s) + g_p^\vee(t^* - s) \right\} &< f_p^\vee \,\overline{\otimes}\, g_p^\vee(t^*) \\ &= f_p^\vee(s^*) + g_p^\vee\,(t^* - s^*) =: z^*,\end{aligned}$$

where either $s^* \in]a_f, b_f]$ or $t^* - s^* \in]a_g, b_g]$.

From $I'_{f_p^\vee} = [T_f, a_f]$, $I'_{g_p^\vee} = [T_g, a_g]$ and Lemma 14.14 it follows that

$$\begin{aligned}I_{f^{-1}_{\downarrow,p}} &= [f(T_f), f(a_f)]\,,\\ I_{g^{-1}_{\downarrow,p}} &= [g(T_g), g(a_g)]\end{aligned}$$

Let us consider now

$$\left(f_p^\vee \,\overline{\otimes}\, g_p^\vee\right)_{\downarrow}^{-1}(z^*) = f^{-1}_{\downarrow,p} \otimes g^{-1}_{\downarrow,p}(z^*).$$

We distinguish two cases: either $z^* \in I_{\otimes_p^{-1}}$, computed according to Equation (F.12), or it is larger than the upper boundary of $I_{\otimes_p^{-1}}$ (as it cannot be less than the lower boundary). In the first case, i.e., $z^* < f(T_f) + g(T_g) + 2 \cdot \operatorname{lcm}(c_f, c_g)$, we have

$$\begin{aligned}f^{-1}_{\downarrow,p} \otimes g^{-1}_{\downarrow,p}(z^*) &= \inf_{0 \le v \le z^*} \left\{ f^{-1}_{\downarrow,p}(v) + g^{-1}_{\downarrow,p}(z^* - v) \right\} \\ &= f^{-1}_{\downarrow,p}(v^*) + g^{-1}_{\downarrow,p}\,(z^* - v^*)\,,\end{aligned}$$

where $v^* \in I'_{f^{-1}_{\downarrow,p}} = [f(T_f), f(a_f)]$, $z^* - v^* \in I'_{g^{-1}_{\downarrow,p}} = [g(T_g), g(a_g)]$ such that the infimum is attained. These elements exist since the lower pseudoinverses are left-continuous over their respective intervals (Lemma 14.8), thus their (min,+) convolution can always be attained (Proposition 14.17). Moreover, since $I'_{f^{-1}_{\downarrow,p}}$ and $I_{g^{-1}_{\downarrow,p}}$ are sufficient to compute $f^{-1}_{\downarrow,p} \otimes g^{-1}_{\downarrow,p}(z)$ for any $z \in I_{\otimes_p^{-1}}$ (Corollary 14.33), it follows that $v^* \in I'_{f^{-1}_{\downarrow,p}}$ and $z^* - v^* \in I_{g^{-1}_{\downarrow,p}}$, hence we do not need any value of f and g outside these intervals to perform the computation for z^* in particular. But, since the interval $\left]a_f, b_f\right]$ was not used to compute $I'_{f^{-1}_{\downarrow,p}}$, nor $\left]a_g, b_g\right]$ to compute $I'_{g^{-1}_{\downarrow,p}}$, this is a contradiction to the assumption that $s^* \in \left]a_f, b_f\right]$ or $t^* - s^* \in \left]a_g, b_g\right]$ are needed for the computation of $f_p^{\vee} \overline{\otimes} g_p^{\vee}$.

In the second case $(z^* \geq f(T_f) + g(T_g) + 2 \cdot \operatorname{lcm}(c_f, c_g))$, $f^{-1}_{\downarrow,p} \otimes g^{-1}_{\downarrow,p}(z^*)$ can be computed by applying the UPP property meaning that

$$f^{-1}_{\downarrow,p} \otimes g^{-1}_{\downarrow,p}(z^*) = f^{-1}_{\downarrow,p} \otimes g^{-1}_{\downarrow,p}\left(z^* - k \cdot d_{\otimes_p^{-1}}\right) + k \cdot c_{\otimes_p^{-1}}$$

for $d_{\otimes_p^{-1}}$ and $c_{\otimes_p^{-1}}$ described in Equation (F.10) and Equation (F.11), respectively, and some $k \in \mathbb{N}$ such that $z^* - k \cdot d_{\otimes_p^{-1}} \in I_{\otimes_p^{-1}}$. We can follow for the latter the same reasoning as in the first case, thus having the same contradiction. This concludes the proof. $\square$

Appendix G

Other Proofs

G.1 Equivalence of UA definitions

Proposition G.1. *A function $f \in \mathcal{U}$ is Ultimately Affine (UA) (defined in Definition 3.3) iff there exist $T \in \mathbb{Q}_+, \sigma, \rho \in \mathbb{Q}$ such that either*

$$f(t) = \rho t + \sigma \quad \forall t \geq T \tag{G.1}$$

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

$$f(t) \overset{(3.2)}{=} \left(f\left(T_f^a\right) - \rho_f \cdot T_f^a\right) + \rho_f \cdot t = \sigma + \rho \cdot t.$$

"$\Leftarrow$" Assume f to verify the condition in Equation (G.1). Therefore, assume that $f(t) = \rho t + \sigma$ for all $t \geq T$. Define $T_f^a := T$, $\rho_f := \rho$. Then for all $t \geq T_f^a$

$$\begin{aligned}
f(t) &= f\left((t - T_f^a) + T_f^a\right) \\
&\overset{(G.1)}{=} \sigma + \rho_f\left((t - T_f^a) + T_f^a\right) \\
&= \left(\rho_f T_f^a + \sigma\right) + \rho_f \cdot (t - T_f^a) \\
&\overset{(G.1)}{=} f(T_f^a) + \rho_f(t - T_f^a).
\end{aligned}$$

This concludes the proof. □

G.2 Subadditivity and superadditivity checks

First, we recall the definition of f°.

Definition 9.2. Let f be a function of $\mathcal{U}$. Then we define f° as

$$f^\circ(t) := \begin{cases} 0, & \text{if } t = 0, \\ f(t), & \text{otherwise.} \end{cases}$$

Lemma 9.3. *Let f be a function of $\mathcal{U}$ such that $f(0) \geq 0$. Then, f is subadditive if and only if $f^\circ = f^\circ \otimes f^\circ$.*

Proof. First, we show that if $f(t) \leq f(s) + f(t-s)$ for any $0 \leq s \leq t$, then f is sub-additive. Let x, y be arbitrary. We define $t = x + y, s = y$. Then,

$$f(x+y) = f(t) \leq f(s) + f(t-s) = f(y) + f(x+y-y) = f(y) + f(x)$$

and thus, f is subadditive. One can easily show that the reverse is also true, so the two are equivalent.

Then, we show that if $f \otimes f = f$, f is subadditive. We know by assumption that for all $t \geq 0$ and τ such that $0 \leq \tau \leq t$, it holds that

$$f(t) = f \otimes f(t) = \inf_{0 \leq s \leq t} \{f(s) + f(t-s)\} \leq f(\tau) + f(t-\tau).$$

By the previous reasoning, f is subadditive. Finally, we show that if $f(0) \geq 0$ and f° is subadditttive, then f is subadditive as well.

We calculate for any $x, y \in \mathbb{Q}_+$ and f° sub-additive that

$$f(x+y) = \begin{cases} f^\circ(x+y) \leq f^\circ(x) + f^\circ(y) = f(x) + f(y), & \text{if } x \text{ and } y > 0, \\ f(0+y) \leq f(0) + f(y) = f(x) + f(y), & \text{if } x = 0 \text{ and } y > 0, \\ \text{symmetric}, & \text{if } x > 0 \text{ and } y = 0, \\ f(0) \leq f(0) + f(0), & \text{if } x = 0 \text{ and } y = 0. \end{cases}$$

Vice versa, we calculate

$$f^\circ(x+y) = \begin{cases} f(x+y) \leq f(x) + f(y) = f^\circ(x) + f^\circ(y), & \text{if } x \text{ and } y > 0, \\ f^\circ(0+y) = f^\circ(0) + f^\circ(y), & \text{if } x = 0 \text{ and } y > 0, \\ \text{symmetric}, & \text{if } x > 0 \text{ and } y = 0, \\ f^\circ(0) = f^\circ(0) + f^\circ(0), & \text{if } x = 0 \text{ and } y = 0. \end{cases}$$

□

Lemma 9.4. *Let f be a function of $\mathcal{U}$ such that $f(0) \leq 0$. Then, f is superadditive if and only if $f^\circ = f^\circ \overline{\otimes} f^\circ$.*

Proof. The proof follows the same steps of Lemma 9.3. We calculate for any $x, y \in \mathbb{Q}_+$ and f° super-additive that

$$f(x+y) = \begin{cases} f^\circ(x+y) \geq f^\circ(x) + f^\circ(y) = f(x) + f(y), & \text{if } x \text{ and } y > 0, \\ f(0+y) \geq f(0) + f(y) = f(x) + f(y), & \text{if } x = 0 \text{ and } y > 0, \\ \text{symmetric}, & \text{if } x > 0 \text{ and } y = 0, \\ f(0) \geq f(0) + f(0), & \text{if } x = 0 \text{ and } y = 0. \end{cases}$$

Vice versa, we calculate

$$f^\circ(x+y) = \begin{cases} f(x+y) \geq f(x) + f(y) = f^\circ(x) + f^\circ(y), & \text{if } x \text{ and } y > 0, \\ f^\circ(0+y) = f^\circ(0) + f^\circ(y), & \text{if } x = 0 \text{ and } y > 0, \\ \text{symmetric}, & \text{if } x > 0 \text{ and } y = 0, \\ f^\circ(0) = f^\circ(0) + f^\circ(0), & \text{if } x = 0 \text{ and } y = 0. \end{cases}$$

□

G.3 Clarifying steps in Appendix F

The proof of Theorem 10.3 in [Lie17, pp. 68] uses an implication, thus a subset relation, to derive an inequality for a supremum. We clarify this step, showing that it is based on calculating a subset over a larger interval.

Lemma G.2. *For any $s, t, x, y \in \mathbb{R}$, it holds that*

$$\sup \{t \mid f(x) \geq s \text{ and } g(y) \geq t - s\} \leq \sup \{t \mid f(x) + g(y) \geq t\}.$$

Proof. We define

$$A := \{t \mid f(x) \geq s \text{ and } g(y) \geq t - s\}$$

and

$$B := \{t \mid f(x) + g(y) \geq t\}.$$

It holds that

$$f(x) \geq s \text{ and } s + g(y) \geq t \;\Rightarrow\; f(x) + g(y) \geq t.$$

In other words, any t, that satisfies the condition on the left-hand side, automatically satisfies the condition on the right-hand side. In other words, it holds that $A \subseteq B$. This, in turn, means that

$$\sup A \leq \sup B$$

and thus the claim follows. □

Example G.3. Assume that $s = 1$ and $f(x) = g(y) = 2$. Then,

$$\sup\{t \mid f(x) \geq s \text{ and } g(y) \geq t - s\} = \sup\{t \mid 2 \geq 1 \text{ and } 2 \geq t - 1\} = 3,$$

and

$$\sup\{t \mid f(x) + g(y) \geq t\} = \sup\{t \mid 2 + 2 \geq t\} = 4.$$

Lemma G.4. *For any $s, t, x, y \in \mathbb{R}$, it holds that*

$$\inf\{t \mid f(x) \leq s \text{ and } g(y) \leq t - s\} \leq \inf\{t \mid f(x) + g(y) \leq t\}.$$

Proof. We define

$$A := \{t \mid f(x) \leq s \text{ and } g(y) \leq t - s\}$$

and

$$B := \{t \mid f(x) + g(y) \leq t\}.$$

It holds that

$$f(x) \leq s \text{ and } s + g(y) \leq t \Rightarrow f(x) + g(y) \leq t.$$

In other words, any t, that satisfies the condition on the left-hand side, automatically satisfies the condition on the right-hand side. In other words, it holds that $A \subseteq B$. This, in turn, means that

$$\inf A \geq \inf B$$

and thus the claim follows. □

Example G.5. Assume that $s = 2$ and $f(x) = g(y) = 1$. Then,

$$\inf\{t \mid f(x) \leq s \text{ and } g(y) \leq t - s\} = \inf\{t \mid 1 \leq 2 \text{ and } 1 \leq t - 2\} = 3,$$

and

$$\inf\{t \mid f(x) + g(y) \leq t\} = \inf\{t \mid 1 + 1 \leq t\} = 2.$$

Appendix H

System Configurations

H.1 Software configuration

As mentioned, *Nancy* can – and does, by default – run most algorithms in parallel, benefiting from the computational power of multicore systems. Note, however, that to minimize the perturbations and improve the fairness of comparisons, in all experiments whose result we report in this thesis, we used *Nancy* in single-threaded mode (`UseParallelism = false`). This allows us to obtain consistent time measurements, and we verified that the execution times of independent replicas of the same experiment differ by fractions of percentage points – for this reason, confidence intervals are omitted.

Moreover, we used the `BigInteger` version of `Rational` type. This has a performance overhead over using 64-bit integers, but has the distinctive advantage of removing all issues with arithmetic overflow – thus we could trust that all experiments would, eventually, terminate.

Execution times are measured using the `System.Diagnostic.Stopwatch` class. When applying our optimizations, the execution times we measure also include those spent testing our hypotheses (e.g., dominance or asymptotic dominance).

In the experiments, we use parametrized curves, whose parameters are then randomly generated. For example, in Chapters 12, 13 and 15, we use $\beta_{R,\theta,h} = \overline{\beta_{R,\theta} + h}$, where $\beta_{R,\theta}$ is a rate-latency curve, with latency θ and rate R, and h is the ordinate of a constant function. We generate these parameters using `System.Random`, constrained so that computations do not become *too* large when hyperperiod explosion occurs – e.g., with a max value of 1000.

H.2 Hardware configurations

In this thesis, we used experimental results that were run on different machines. For ease of reference, we collect their full specifications here.

System 1. Desktop PC, with Intel Core i9-9900, 16 GB of DRAM @3200 MHz, Windows 10

System 2. Desktop PC, with Intel Core i9-9900, 32 GB of DRAM @3200 MHz, Windows 10

System 3. Desktop PC, with AMD Ryzen 7 5800X, 32 GB of DRAM @3000 MHz, Windows 10

System 4. Laptop PC, with Intel i7-10750H, 32 GB of DDR4 @3000 MHz, Windows 10

System 5. Cloud Virtual Machine, with 24 virtual Intel Xeon Processors (Cascade-Lake) cores @2.2 GHz, 32 GB of DRAM, Ubuntu 22.04

Index of notation

Sets

$\mathbb{R}_+$	Set of non-negative real numbers, includes 0.	
$\mathbb{R}$	Set of real numbers, does not include $\pm\infty$.	
$\mathbb{Q}_+$	Set of non-negative rational numbers, includes 0.	
$\mathbb{Q}$	Set of rational numbers, does not include $\pm\infty$.	
$\mathbb{N}$	Set of natural numbers, does not include 0.	
$\mathbb{N}_0$	Set of non-negative integers, including 0.	
$\mathcal{U}$	Set of ultimately pseudo-periodic, piecewise affine, $\mathbb{Q}_+ \to \mathbb{Q} \cup \{+\infty, -\infty\}$ functions.	Section 3.2

DNC operators

$f \wedge g$	Minimum	Section 2.2
$f \vee g$	Maximum	Section 2.2
$f \otimes g$	(min,+) convolution	Section 2.2
$f \overline{\otimes} g$	(max,+) convolution	Section 2.2
$\overline{f}$	Subadditive closure	Definition 2.1
$\overline{\overline{f}}$	Superadditive closure	Definition 2.5
$f_{\downarrow}^{-1}$	Lower pseudoinverse	Definition 4.9
$f_{\uparrow}^{-1}$	Upper pseudoinverse	Definition 4.9
$f \circ g$	Composition, $f(g(t))$	Chapter 11

UPP functions

T_f	Pseudo-period start of f	Definition 3.1
d_f	Pseudo-period length of f	Definition 3.1
c_f	Pseudo-period height of f	Definition 3.1
S_f^I	Sequence that represents f over interval I	Definition 3.6
R_f	Tuple (S, T, d, c) that represents f over $\mathbb{Q}_+$	Section 3.2
$n(S)$	Number of elements included in sequence S	Section 3.2
$f\|_I^\wedge$	$f\|_I^\wedge(t) = f(t)$ if $t \in I$, $+\infty$ otherwise	Definition 3.8
$f\|_I^\vee$	$f\|_I^\vee(t) = f(t)$ if $t \in I$, $-\infty$ otherwise	Definition 3.8
$f_t^\wedge \wedge f_p^\wedge$	Min decomposition of f over its transient and periodic parts.	Equation (3.3)
$f_t^\vee \wedge f_p^\vee$	Max decomposition of f over its transient and periodic parts.	Equation (3.3)

Operators *over interval*

$f_{\downarrow,I}^{-1}$	Lower pseudoinverse over interval I	Definition 14.6
$f_{\uparrow,I}^{-1}$	Upper pseudoinverse over interval I	Definition 14.6
$f_{\downarrow,p}^{-1}$	Lower pseudoinverse over interval $\left[T_f, +\infty\right[$	Definition 14.6
$f_{\uparrow,p}^{-1}$	Upper pseudoinverse over interval $\left[T_f, +\infty\right[$	Definition 14.6
$[f]_a$	Reconstruction operator	Equation (14.19)

Other notations

$[a]^+$	Positive part, equal to $\max(a, 0)$.	
f°	$f^\circ(0) = 0, f^\circ(t) = f(t)\ \forall\, t > 0$	Definition 9.2
$\beta_{r,l}$	Rate-latency curve with rate r and latency l.	Section 2.1
$\lfloor \beta \rfloor_{rl}$	Rate-latency lower bound for β	Chapter 6

Publications

Conference proceedings

[ACSZ20] Matteo Andreozzi, Frances Conboy, Giovanni Stea, and Raffaele Zippo. “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), Invited Paper*. IEEE. 2020, pp. 79–86.

[RSLSSZZAH21] Falk Rehm, Jörg Seitter, Jan-Peter Larsson, Selma Saidi, Giovanni Stea, Raffaele Zippo, Dirk Ziegenbein, Matteo Andreozzi, and Arne Hamann. “The Road towards Predictable Automotive High-Performance Platforms”. In: *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2021, pp. 1915–1924.

[ZNS23a] Raffaele Zippo, Paul Nikolaus, and Giovanni Stea. “Isospeed: Improving (min,+) Convolution by Exploiting (min,+)/(max,+) Isomorphism”. In: *35th Euromicro Conference on Real-Time Systems*. 2023.

Journal articles

[AFGSZ22] Matteo Andreozzi, Antonio Frangioni, Laura Galli, Giovanni Stea, and Raffaele Zippo. “A MILP approach to DRAM access worst-case analysis”. In: *Computers & Operations Research* 143 (2022).

[ZNS23c] Raffaele Zippo, Paul Nikolaus, and Giovanni Stea. “Extending the Network Calculus Algorithmic Toolbox for Ultimately Pseudo-Periodic Functions: Pseudo-Inverse and Composition”. In: *Discrete Event Dynamic Systems* (Accepted December 2022, Yet to appear as of July 2023). DOI: `10.1007/s10626-022-00373-5`.

[ZS22] Raffaele Zippo and Giovanni Stea. “Nancy: An efficient parallel Network Calculus library”. In: *SoftwareX* 19 (2022). DOI: `https://doi.org/10.1016/j.softx.2022.101178`. URL: `https://www.sciencedirect.com/science/article/pii/S235271102200108X`.

[ZS23] Raffaele Zippo and Giovanni Stea. “Computationally efficient worst-case analysis of flow-controlled networks with network calculus”. In: *IEEE Transactions on Information Theory* 69.4 (2023), pp. 2664–2690.

Patents

[ACSZ22] Matteo Maria Andreozzi, Michael Andrew Campbell, Giovanni Stea, and Raffaele Zippo. “Method, system and device for electronic interconnect delay bound determination”. US Patent 11,455,268. Sept. 2022.

Artifacts

[ZNS23b] Raffaele Zippo, Paul Nikolaus, and Giovanni Stea. “Isospeed: Improving (min,+) Convolution by Exploiting (min,+)/(max,+) Isomorphism (Artifact)”. In: *Dagstuhl Artifacts Series* 9 (2023).

Software

[ZSd] Raffaele Zippo and Giovanni Stea. *Nancy source repository on GitHub*. URL: `https://github.com/rzippo/nancy`.

Talks

Nancy: a library for Network Calculus, Best Student Presentation Award (ex-aequo), 6th Workshop on Network Calculus, 2022.

Algebraic transformations for network paths with hop-by-hop flow control, 5th Workshop on Network Calculus, 2020.

Bibliography

[ACOR99] Rajeev Agrawal, Rene L Cruz, Clayton Okino, and Rajendran Rajan. "Performance bounds for flow control protocols". In: *IEEE/ACM transactions on networking* 7.3 (1999), pp. 310–323.

[ACSZ20] Matteo Andreozzi, Frances Conboy, Giovanni Stea, and Raffaele Zippo. "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), Invited Paper*. IEEE. 2020, pp. 79–86.

[AFGSZ22] Matteo Andreozzi, Antonio Frangioni, Laura Galli, Giovanni Stea, and Raffaele Zippo. "A MILP approach to DRAM access worst-case analysis". In: *Computers & Operations Research* 143 (2022).

[Bal11] Simonetta Balsamo. "Queueing Networks with Blocking: Analysis, Solution Algorithms and Properties". In: *Network Performance Engineering: A Handbook on Convergent Multi-Service Networks and Next Generation Internet*. Ed. by Demetres D. Kouvatsos. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 233–257. ISBN: 978-3-642-02742-0. DOI: 10.1007/978-3-642-02742-0_11. URL: https://doi.org/10.1007/978-3-642-02742-0_11.

[BBL18] Anne Bouillard, Marc Boyer, and Euriell Le Corronc. *Deterministic Network Calculus: From Theory to Practical Implementation*. Hoboken, NJ: Wiley, 2018.

[BCGHLL09] Anne Bouillard, Bertrand Cottenceau, Bruno Gaujal, Laurent Hardouin, Sébastien Lagrange, and Mehdi Lhommeau. "COINC Library: A Toolbox for the Network Calculus: Invited Presentation, Extended Abstract". In: *Proceedings of the Fourth International ICST Conference on Performance Evaluation Methodologies and Tools*. VALUETOOLS '09. Pisa, Italy: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2009. ISBN: 9789639799707.

[BCOQ92] François Baccelli, Guy Cohen, Geert Jan Olsder, and Jean-Pierre Quadrat. “Synchronization and linearity: an algebra for discrete event systems”. In: (1992).

[BD22] Marc Boyer and Hugo Daigmorte. “Improved service curve for element with known transmission rate”. In: *IEEE Networking Letters* (2022), pp. 1–1. DOI: 10.1109/LNET.2022.3150649.

[BGDM20] Marc Boyer, Amaury Graillat, Benot Dupont De Dinechin, and Jörn Migge. “Bounding the delays of the MPPA network-on-chip with network calculus: Models and benchmarks”. In: *Performance Evaluation* 143 (2020), p. 102124.

[BJLL06] Amit Bose, Xiaoyue Jiang, Bin Liu, and Gang Li. “Analysis of manufacturing blocking systems with network calculus”. In: *Performance Evaluation* 63.12 (2006), pp. 1216–1234.

[BJT09] Anne Bouillard, Laurent Jouhet, and Eric Thierry. “Service curves in Network Calculus: dos and don’ts”. PhD thesis. INRIA, 2009.

[BJT10] Anne Bouillard, Laurent Jouhet, and Eric Thierry. “Tight Performance Bounds in the Worst-Case Analysis of Feed-Forward Networks”. In: *2010 Proceedings IEEE INFOCOM*. 2010, pp. 1–9. DOI: 10.1109/INFCOM.2010.5461912.

[BLMS10] Luca Bisti, Luciano Lenzini, Enzo Mingozzi, and Giovanni Stea. “Deborah: A tool for worst-case analysis of FIFO tandems”. In: *International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*. Springer. 2010, pp. 152–168.

[BLMS12] Luca Bisti, Luciano Lenzini, Enzo Mingozzi, and Giovanni Stea. “Numerical analysis of worst-case end-to-end delay bounds in FIFO tandem networks”. In: *Real-Time Systems* 48.5 (2012), pp. 527–569.

[Bou21] Anne Bouillard. “Individual Service Curves for Bandwidth-Sharing Policies Using Network Calculus”. In: *IEEE Networking Letters* 3.2 (2021), pp. 80–83. DOI: 10.1109/LNET.2021.3067766.

[Boy10] Marc Boyer. “NC-maude: a rewriting tool to play with network calculus”. In: *International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*. Springer. 2010, pp. 137–151.

[BPC09] Anne Bouillard, Linh T. X. Phan, and Samarjit Chakraborty. “Lightweight Modeling of Complex State Dependencies in Stream Processing Systems”. In: *Proceedings of the 2009 15th IEEE Symposium on Real-Time and Embedded Technology and Applications*. RTAS ’09. USA: IEEE Computer Society, 2009, pp. 195–204. ISBN: 9780769536361. DOI: 10.1109/RTAS.2009.27. URL: https://doi.org/10.1109/RTAS.2009.27.

[BRD22] Marc Boyer, Pierre Roux, and Hugo Daigmorte. “Checking validity of the min-plus operations involved in the analysis of a real-time embedded network”. In: *ERTS 2022-11th European Congress Embedded Real Time System*. 2022.

[BS12] Anne Bouillard and Giovanni Stea. “Exact worst-case delay for FIFO-multiplexing tandems”. In: *6th International ICST Conference on Performance Evaluation Methodologies and Tools*. IEEE. 2012, pp. 158–167.

[BS14] Steffen Bondorf and Jens B. Schmitt. “The DiscoDNC v2 – A Comprehensive Tool for Deterministic Network Calculus”. In: *Proceedings of the International Conference on Performance Evaluation Methodologies and Tools*. VALUETOOLS ’14. Dec. 2014, pp. 44–49. URL: https://dl.acm.org/citation.cfm?id=2747659.

[BS15] Anne Bouillard and Giovanni Stea. “Exact worst-case delay in FIFO-multiplexing feed-forward networks”. In: *IEEE/ACM Transactions on Networking (TON)* 23.5 (2015), pp. 1387–1400.

[BS17] Michael Beck and Jens Schmitt. “Generalizing window flow control in bivariate network calculus to enable leftover service in the loop”. In: *Performance Evaluation* 114 (2017), pp. 45–55. ISSN: 0166-5316. DOI: https://doi.org/10.1016/j.peva.2017.04.008. URL: https://www.sciencedirect.com/science/article/pii/S0166531616301808.

[BSF09] Henri Bauer, Jean-Luc Scharbarg, and Christian Fraboul. “Applying and optimizing trajectory approach for performance evaluation of AFDX avionics network”. In: *2009 IEEE Conference on Emerging Technologies & Factory Automation*. IEEE. 2009, pp. 1–8.

[BSF10] Henri Bauer, Jean-Luc Scharbarg, and Christian Fraboul. “Worst-case end-to-end delay analysis of an avionics AFDX network”. In: *2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)*. IEEE. 2010, pp. 1220–1224.

[BSGE09] Mohamed Bakhouya, S Suboh, Jaafar Gaber, and T El-Ghazawi. "Analytical modeling and evaluation of on-chip interconnects using network calculus". In: *2009 3rd ACM/IEEE International Symposium on Networks-on-Chip*. IEEE. 2009, pp. 74–79.

[BSS12a] Marc Boyer, Giovanni Stea, and William Mangoua Sofack. "Deficit Round Robin with network calculus". In: *6th International ICST Conference on Performance Evaluation Methodologies and Tools*. 2012, pp. 138–147. DOI: 10.4108/valuetools.2012.250202.

[BSS12b] Marc Boyer, Giovanni Stea, and William Mangoua Sofack. "Deficit Round Robin with network calculus". In: *6th International ICST Conference on Performance Evaluation Methodologies and Tools, Cargese, Corsica, France, October 9-12, 2012*. 2012, pp. 138–147. DOI: 10.4108/valuetools.2012.250202. URL: https://doi.org/10.4108/valuetools.2012.250202.

[BT07] Anne Bouillard and Eric Thierry. *An Algorithmic Toolbox for Network Calculus*. Research Report RR-6094. INRIA, 2007, p. 44. URL: https://hal.inria.fr/inria-00123643.

[BT08] Anne Bouillard and Éric Thierry. "An algorithmic toolbox for network calculus". In: *Discrete Event Dynamic Systems* 18.1 (2008), pp. 3–49.

[BT16] Anne Bouillard and Eric Thierry. "Tight performance bounds in the worst-case analysis of feed-forward networks". In: *Discrete Event Dynamic Systems* 26.3 (2016), pp. 383–411. DOI: 10.1007/s10626-015-0213-2. URL: https://doi.org/10.1007/s10626-015-0213-2.

[CBL06] Florin Ciucu, Almut Burchard, and Jörg Liebeherr. "Scaling properties of statistical end-to-end bounds in the network calculus". In: *IEEE Transactions on Information Theory* 52.6 (2006), pp. 2300–2312.

[Cha00] Cheng-Shang Chang. *Performance guarantees in communication networks*. New York, USA: Springer-Verlang, 2000.

[Cha97] Cheng-shang Chang. "On Deterministic Traffic Regulation and Service Guarantees: A Systematic Approach by Filtering". In: *IEEE Transactions on Information Theory* 44 (1997), pp. 1096–1107.

[CNS22a] Vlad-Cristian Constantin, Paul Nikolaus, and Jens Schmitt. “Improving Performance Bounds for Weighted Round-Robin Schedulers under Constrained Cross-Traffic”. In: *Proceedings IFIP Networking 2022 Conference (NETWORKING’22)*. ISBN 978-3-903176-48-5. Catania, Italy: IEEE, June 2022. DOI: `10.23919/IFIPNetworking55013.2022.9829772`. URL: `/discofiles/publicationsfiles/CNS22-1.pdf`.

[CNS22b] Vlad-Cristian Constantin, Paul Nikolaus, and Jens Schmitt. “Original and Erratum: Improving Performance Bounds for Weighted Round-Robin Schedulers under Constrained Cross-Traffic”. In: arXiv, Dec. 2022. DOI: `10.48550/ARXIV.2202.08381`. URL: `/discofiles/publicationsfiles/CNS22-1_erratum.pdf`.

[Cru91a] Rene L Cruz. “A calculus for network delay, part I: Network elements in isolation”. In: *IEEE Transactions on information theory* 37.1 (1991), pp. 114–131.

[Cru91b] Rene L Cruz. “A calculus for network delay, part II: Network analysis”. In: *IEEE Transactions on information theory* 37.1 (1991), pp. 132–141.

[CS12] Florin Ciucu and Jens Schmitt. “Perspectives on network calculus: no free lunch, but still good value”. In: *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*. 2012, pp. 311–322.

[CSEF06] Hussein Charara, J-L Scharbarg, Jérôme Ermont, and Christian Fraboul. “Methods for bounding end-to-end delays on an AFDX network”. In: *18th Euromicro Conference on Real-Time Systems (ECRTS’06)*. IEEE. 2006, 10–pp.

[FE17] HE Feng and LI Ershuai. “Deterministic bound for avionics switched networks according to networking features using network calculus”. In: *Chinese Journal of Aeronautics* 30.6 (2017), pp. 1941–1957.

[Fid06] Markus Fidler. “An end-to-end probabilistic network calculus with moment generating functions”. In: *200614th IEEE International Workshop on Quality of Service*. IEEE. 2006, pp. 261–270.

[GC21] Liang Guo and Paul Congdon. “IEEE 802 Nendica Report: Intelligent Lossless Data Center Networks”. In: *IEEE SA Industry Connections–IEEE 802 Nendica Report: Intelligent Lossless Data Center Networks* (2021), pp. 1–44.

[GM18] Frédéric Giroudot and Ahlem Mifdaoui. "Buffer-aware worst-case timing analysis of wormhole NoCs using network calculus". In: *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE. 2018, pp. 37–48.

[GM19] Frederic Giroudot and Ahlem Mifdaoui. "Tightness and computation assessment of worst-case delay bounds in wormhole networks-on-chip". In: *Proceedings of the 27th International Conference on Real-Time Networks and Systems*. 2019, pp. 19–29.

[GSSAA19] Prateesh Goyal, Preey Shah, Naveen Kr. Sharma, Mohammad Alizadeh, and Thomas E. Anderson. "Backpressure Flow Control". In: *Proceedings of the 2019 Workshop on Buffer Sizing*. BS '19. Palo Alto, CA, USA: Association for Computing Machinery, 2019. ISBN: 9781450377454. DOI: `10.1145/3375235.3375239`. URL: `https://doi.org/10.1145/3375235.3375239`.

[GY13] Nan Guan and Wang Yi. "Finitary real-time calculus: Efficient performance analysis of distributed embedded systems". In: *2013 IEEE 34th Real-Time Systems Symposium*. 2013, pp. 330–339.

[IEEE754] "IEEE Standard for Floating-Point Arithmetic". In: *IEEE Std 754-2019 (Revision of IEEE 754-2008)* (2019), pp. 1–84. DOI: `10.1109/IEEESTD.2019.8766229`.

[JL08] Yuming Jiang and Yong Liu. *Stochastic network calculus*. Vol. 1. Springer, 2008.

[KGNMP18] Anders Ellersgaard Kalør, Rene Guillaume, Jimmy Jessen Nielsen, Andreas Mueller, and Petar Popovski. "Network slicing in industry 4.0 applications: Abstraction methods and end-to-end analysis". In: *IEEE Transactions on Industrial Informatics* 14.12 (2018), pp. 5419–5427.

[LBS16] Kai Lampka, Steffen Bondorf, and Jens Schmitt. "Achieving efficiency without sacrificing model accuracy: Network calculus on compact domains". In: *2016 IEEE 24th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE. 2016, pp. 313–318.

[LBSGY17] Kai Lampka, Steffen Bondorf, Jens B. Schmitt, Nan Guan, and Wang Yi. "Generalized Finitary Real-Time Calculus". In: *Proceedings of the 36th IEEE International Conference on Computer Communications (INFOCOM 2017)*. 2017.

[LCH14] Euriell Le Corronc, Bertrand Cottenceau, and Laurent Hardouin. “Container of (min,+)-linear systems”. In: *Discrete Event Dynamic Systems* 24.1 (2014), pp. 15–52.

[LHL17a] Natchanon Luangsomboon, Robert Hesse, and Jorg Liebeherr. “Fast Min-plus Convolution and Deconvolution on GPUs”. In: *Proceedings of the 11th EAI International Conference on Performance Evaluation Methodologies and Tools*. VALUETOOLS 2017. Venice, Italy: Association for Computing Machinery, 2017, pp. 126–131. ISBN: 9781450363464. DOI: `10.1145/3150928.3150958`. URL: `https://doi.org/10.1145/3150928.3150958`.

[LHL17b] Natchanon Luangsomboon, Robert Hesse, and Jorg Liebeherr. “Fast Min-plus Convolution and Deconvolution on GPUs”. In: *Proceedings of the 11th EAI International Conference on Performance Evaluation Methodologies and Tools*. VALUETOOLS 2017. Venice, Italy: Association for Computing Machinery, 2017, pp. 126–131. ISBN: 9781450363464. DOI: `10.1145/3150928.3150958`. URL: `https://doi.org/10.1145/3150928.3150958`.

[Lie17] Jörg Liebeherr. “Duality of the max-plus and min-plus network calculus”. In: *Foundations and Trends in Networking* 11.3-4 (2017), pp. 139–282. ISSN: 15540588. DOI: `10.1561/1300000059`.

[LMMS06] Luciano Lenzini, Linda Martorini, Enzo Mingozzi, and Giovanni Stea. “Tight end-to-end per-flow delay bounds in FIFO multiplexing sink-tree networks”. In: *Performance Evaluation* 63.9-10 (2006), pp. 956–987.

[LT01] Jean-Yves Le Boudec and Patrick Thiran. *Network calculus: a theory of deterministic queuing systems for the internet*. Berlin, Germany: Springer Science & Business Media, 2001.

[MHG20] Lisa Maile, Kai-Steffen Hielscher, and Reinhard German. “Network Calculus Results for TSN: An Introduction”. In: *2020 Information Communication Technologies Conference (ICTC)*. IEEE. 2020, pp. 131–140.

[MSB19] Ehsan Mohammadpour, Eleni Stai, and Jean-Yves Le Boudec. “Improved delay bound for a service curve element with known transmission rate”. In: *IEEE Networking Letters* 1.4 (2019), pp. 156–159.

[MSB22] Ehsan Mohammadpour, Eleni Stai, and Jean-Yves Le Boudec. “Improved Network Calculus Delay Bounds in Time-Sensitive Networks”. In: *arXiv preprint arXiv:2204.10906* (2022).

[PLSK11] Victor Pollex, Henrik Lipskoch, Frank Slomka, and Steffen Kollmann. “Runtime improved computation of path latencies with the real-time calculus”. In: *Proceedings of the 1st International Workshop on Worst-Case Traversal Time*. 2011, pp. 58–65.

[PS20] Vitaly G. Promyslov and Kirill V. Semenkov. “Assessment of deterministic delay bounds for a DoS-attack prevention device with a static window flow control”. In: *IFAC-PapersOnLine* 53.2 (2020). 21st IFAC World Congress, pp. 11089–11093. ISSN: 2405-8963. DOI: `https://doi.org/10.1016/j.ifacol.2020.12.251`. URL: `https://www.sciencedirect.com/science/article/pii/S2405896320305280`.

[PS22] Victor Pollex and Frank Slomka. “A Mathematical Comparison Between Response-Time Analysis and Real-Time Calculus for Fixed-Priority Preemptive Scheduling”. In: *34th Euromicro Conference on Real-Time Systems (ECRTS 2022)*. Ed. by Martina Maggio. Vol. 231. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022, 7:1–7:25. ISBN: 978-3-95977-239-6. DOI: `10.4230/LIPIcs.ECRTS.2022.7`. URL: `https://drops.dagstuhl.de/opus/volltexte/2022/16324`.

[QLD09] Yue Qian, Zhonghai Lu, and Wenhua Dou. “Analysis of worst-case delay bounds for best-effort communication in wormhole networks on chip”. In: *Networks-on-Chip, 2009. NoCS 2009. 3rd ACM/IEEE International Symposium on*. IEEE. 2009, pp. 44–53.

[QLD10] Yue Qian, Zhonghai Lu, and Wenhua Dou. “Analysis of worst-case delay bounds for on-chip packet-switching networks”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 29.5 (2010), pp. 802–815.

[RQB22] Pierre Roux, Sophie Quinton, and Marc Boyer. “A formal link between response time analysis and network calculus”. In: *34th Euromicro Conference on Real-Time Systems ECRTS 2022*. 2022.

[RRB21] Lucien Rakotomalala, Pierre Roux, and Marc Boyer. “Verifying Min-Plus Computations with Coq”. In: *NASA Formal Methods*. Ed. by Aaron Dutle, Mariano M. Moscato, Laura Titolo, César A. Muñoz, and Ivan Perez. Cham: Springer International Publishing, 2021, pp. 287–303. ISBN: 978-3-030-76384-8.

[RSLSSZZAH21] Falk Rehm, Jörg Seitter, Jan-Peter Larsson, Selma Saidi, Giovanni Stea, Raffaele Zippo, Dirk Ziegenbein, Matteo Andreozzi, and Arne Hamann. "The Road towards Predictable Automotive High-Performance Platforms". In: *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2021, pp. 1915–1924.

[RTaWa] RealTime-at-Work. *Minplus-Console V1.5.3 User Manual*. URL: `https://www.realtimeatwork.com/minplus-console/RTaW-MinplusConsole-UserManual.pdf` (visited on 01/09/2023).

[RTaWb] RealTime-at-Work. *Online Min-Plus Interpreter for Network Calculus*. URL: `http://realtimeatwork.com/minplus-playground` (visited on 11/01/2021).

[RTaWc] RealTime-at-Work. *RTaW-Pegase (min,+) library*. URL: `https://www.realtimeatwork.com/rtaw-pegase-libraries/` (visited on 04/05/2022).

[SB17] Philipp Schon and Steffen Bondorf. "Towards Unified Tool Support for Real-time Calculus & Deterministic Network Calculus". In: *Proceedings of the Euromicro Conference on Real-Time Systems, Work-in-Progress Session*. ECRTS '17. June 2017.

[SBP17] Jens Schmitt, Steffen Bondorf, and Wint Yi Poe. "The sensor network calculus as key to the design of wireless sensor networks with predictable performance". In: *Journal of Sensor and Actuator Networks* 6.3 (2017), p. 21.

[SLSF18] Aakash Soni, Xiaoting Li, Jean-Luc Scharbarg, and Christian Fraboul. "Optimizing Network Calculus for Switched Ethernet Network with Deficit Round Robin". In: *2018 IEEE Real-Time Systems Symposium (RTSS)*. 2018, pp. 300–311. DOI: `10.1109/RTSS.2018.00046`.

[SPLT10] Urban Suppiger, Simon Perathoner, Kai Lampka, and Lothar Thiele. *A simple approximation method for reducing the complexity of Modular Performance Analysis*. TIK-Report 329. Computer Engineering and Networks Laboratory Swiss Federal Institute of Technology (ETH): Computer Engineering and Networks Laboratory – Swiss Federal Institute of Technology (ETH), Aug. 2010.

[SR05] Jens B Schmitt and Utz Roedig. "Sensor network calculus–a framework for worst case analysis". In: *International Conference on Distributed Computing in Sensor Systems*. Springer. 2005, pp. 141–154.

[SSB22] Alexander Scheffler, Jens Schmitt, and Steffen Bondorf. “Searching for Upper Delay Bounds in FIFO Multiplexing Feedforward Networks”. In: *Proceedings of the 30th International Conference on Real-Time Networks and Systems*. 2022, pp. 230–241.

[SSH07] Henrik Schioler, Hans P Schwefel, and Martin B Hansen. “Cync: a matlab/simulink toolbox for network calculus”. In: *2nd International ICST Conference on Performance Evaluation Methodologies and Tools*. 2007.

[SZF08] Jens B Schmitt, Frank A Zdarsky, and Markus Fidler. “Delay bounds under arbitrary multiplexing: When network calculus leaves you in the lurch...” In: *IEEE INFOCOM 2008-The 27th Conference on Computer Communications*. IEEE. 2008, pp. 1669–1677.

[SZT07] Jens B Schmitt, Frank A Zdarsky, and Lothar Thiele. “A comprehensive worst-case calculus for wireless sensor networks with in-network processing”. In: *28th IEEE International Real-Time Systems Symposium (RTSS 2007)*. IEEE. 2007, pp. 193–202.

[TB21] Seyed Mohammadhossein Tabatabaee and Jean-Yves Le Boudec. “Deficit Round-Robin: A Second Network Calculus Analysis”. In: *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 2021, pp. 171–183. DOI: 10.1109/RTAS52030.2021.00022.

[TCN00] Lothar Thiele, Samarjit Chakraborty, and Martin Naedele. “Real-time calculus for scheduling hard real-time systems”. In: *2000 IEEE International Symposium on Circuits and Systems (ISCAS)*. Vol. 4. IEEE. 2000, pp. 101–104.

[TL22] Seyed Mohammadhossein Tabatabaee and Jean-Yves Le Boudec. “Deficit Round-Robin: A Second Network Calculus Analysis”. In: *IEEE/ACM Transactions on Networking* (2022).

[TLB21] Seyed Mohammadhossein Tabatabaee, Jean-Yves Le Boudec, and Marc Boyer. “Interleaved weighted round-robin: A network calculus analysis”. In: *IEICE Transactions on Communications* 104.12 (2021), pp. 1479–1493.

[Wan06] Ernesto Wandeler. “Modular performance analysis and interface-based design for embedded real-time systems”. PhD thesis. ETH Zurich, 2006.

[WCHLL21] Shie-Yuan Wang, Yo-Ru Chen, Hsien-Chueh Hsieh, Ruei-Syun Lai, and Yi-Bing Lin. "A Flow Control Scheme Based on Per Hop and Per Flow in Commodity Switches for Lossless Networks". In: *IEEE Access* 9 (2021), pp. 156013–156029. DOI: 10.1109/ACCESS.2021.3129595.

[WTa] Ernesto Wandeler and Lothar Thiele. *Real-Time Calculus (RTC) Toolbox*. URL: http://www.mpa.ethz.ch/Rtctoolbox (visited on 12/17/2022).

[WTb] Ernesto Wandeler and Lothar Thiele. *RTC Toolbox Tutorial*. URL: https://www.mpa.ethz.ch/tutorial/Tutorial.html (visited on 12/17/2022).

[ZCWW19] Jiayi Zhang, Lihao Chen, Tongtong Wang, and Xinyuan Wang. "Analysis of TSN for industrial automation based on network calculus". In: *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE. 2019, pp. 240–247.

[ZHLC20] Boyang Zhou, Isaac Howenstine, Siraphob Limprapaipong, and Liang Cheng. "A Survey on Network Calculus Tools for Network Infrastructure in Real-Time Systems". In: *IEEE Access* 8 (2020), pp. 223588–223605. DOI: 10.1109/ACCESS.2020.3043600.

[ZNS23a] Raffaele Zippo, Paul Nikolaus, and Giovanni Stea. "Isospeed: Improving (min,+) Convolution by Exploiting (min,+)/(max,+) Isomorphism". In: *35th Euromicro Conference on Real-Time Systems*. 2023.

[ZNS23b] Raffaele Zippo, Paul Nikolaus, and Giovanni Stea. "Isospeed: Improving (min,+) Convolution by Exploiting (min,+)/(max,+) Isomorphism (Artifact)". In: *Dagstuhl Artifacts Series* 9 (2023).

[ZNS23c] Raffaele Zippo, Paul Nikolaus, and Giovanni Stea. "Extending the Network Calculus Algorithmic Toolbox for Ultimately Pseudo-Periodic Functions: Pseudo-Inverse and Composition". In: *Discrete Event Dynamic Systems* (Accepted December 2022, Yet to appear as of July 2023). DOI: 10.1007/s10626-022-00373-5.

[ZPZDB21] Luxi Zhao, Paul Pop, Zhong Zheng, Hugo Daigmorte, and Marc Boyer. "Latency Analysis of Multiple Classes of AVB Traffic in TSN With Standard Credit Behavior Using Network Calculus". In: *IEEE Trans. Ind. Electron.* 68.10 (2021), pp. 10291–10302. DOI: 10.1109/TIE.2020.3021638. URL: https://doi.org/10.1109/TIE.2020.3021638.

[ZSa] Raffaele Zippo and Giovanni Stea. *Example of non-terminating computation using RTC Toolbox*. URL: https://gist.github.com/rzippo/8a7575b5517db8d9f871171f6ce08ef9.

[ZSb] Raffaele Zippo and Giovanni Stea. *Nancy library for Network Calculus*. URL: http://nancy.unipi.it/.

[ZSc] Raffaele Zippo and Giovanni Stea. *Nancy packages on NuGet*. URL: https://www.nuget.org/packages/Unipi.Nancy/.

[ZSd] Raffaele Zippo and Giovanni Stea. *Nancy source repository on GitHub*. URL: https://github.com/rzippo/nancy.

[ZS22] Raffaele Zippo and Giovanni Stea. "Nancy: An efficient parallel Network Calculus library". In: *SoftwareX* 19 (2022). DOI: https://doi.org/10.1016/j.softx.2022.101178. URL: https://www.sciencedirect.com/science/article/pii/S235271102200108X.

[ZS23] Raffaele Zippo and Giovanni Stea. "Computationally efficient worst-case analysis of flow-controlled networks with network calculus". In: *IEEE Transactions on Information Theory* 69.4 (2023), pp. 2664–2690.

[ZYD11] Lianming Zhang, Jianping Yu, and Xiaoheng Deng. "Modelling the guaranteed QoS for wireless sensor networks: a network calculus approach". In: *EURASIP Journal on Wireless Communications and Networking* 2011.1 (2011), pp. 1–14.

Colophon

This document was typeset in LaTeX using the template provided by University of Florence. Code listings were typeset using *minted*; plots were typeset using *TikZ* and *pgfplots*; diagrams are produced with *draw.io* (now *diagrams.net*) and the custom package *drawio.sty* (hosted on GitHub repository drawio-latex).