Parallel reversal schedules using more checkpoints than processors

Diploma Thesis

HUMBOLDT UNIVERSITY OF BERLIN
FACULTY OF MATHEMATICS AND NATURAL SCIENCES
DEPARTMENT OF MATHEMATICS

Presentor:	Volker Diels-Grabsch
Supervisor:	Prof. Andreas Griewank
Second assessor:	Prof. Andrea Walther
Submission:	Berlin, September 24, 2015

Abstract

Parallel reversal schedules describe how to calculate the states of an evolutionary system, such as atmospheric and oceanographic simulations, in reverse order without having to keep all states in memory. This is possible without any increase in computation time, by recalculating intermediate results using multiple processors on a parallel computer system. These schedules are not only applied physical simulations that need to run in reverse, but also in algorithmic differentiation, which in turn is used, among others, in nonlinear optimization and to solve partial differential equations. Earlier research led to optimal schedules under the central assumption that if k states can be kept in memory, there are enough processors to run computations on roughly half of them in parallel, while the other half can only be used for holding checkpoints.

   This diploma thesis is an attempt to continue the research by relaxing the central assumption, such that memory for a large number of plain checkpoints can be used with a comparatively small number of processors. To cope with this challenge, a symbolic approach to reversal schedules is proposed, and a comprehensive algebra is developed to analyze schedules via their profiles. This algebra is very generic and could have applications outside parallel reversal schedules. Using that instrument, new optimal schedules are developed, to be applied in situations where the earlier schedules are not applicable. Moreover, suboptimal schedules are provided where optimal schedules could not be found in a systematic way.

Acknowledgments

First of all, I would like to thank Prof. Andreas Griewank for providing the opportunity to write my thesis on this exciting topic, for his patience, for guiding me through the research and writing phases, and most of all, for accepting me as his last diploma degree candidate before leaving the Humboldt University of Berlin.

I would also like to thank Prof. Andrea Walther not just for her solid dissertation on which this thesis is based, but also for her advice and support whenever I needed it.

Moreover, I appreciate the advice given by Dr. Louchka Popova-Zeugmann on my Petri net drafts.

I am particularly grateful for the continuous motivation given by Berit Grußien, who also read drafts and gave valuable hints to bring this thesis into shape.

Finally, and most importantly, I am deeply indebted to my wife Antje for her unfailing support during this intensive phase of my life.

Contents

6

1. Introduction

1.1. Reversal of Evolutionary Systems

Evolutionary systems are mathematical models that can be described as a sequence of states

$$s_0, s_1, s_2, \ldots, s_\ell$$

where each state can be calculated directly from the previous state. That is, there are functions $F_1, F_2, \ldots$ such that:

$$s_1 = F_1(s_0)$$
$$s_2 = F_2(s_1)$$
$$\ldots$$
$$s_\ell = F_\ell(s_{\ell-1})$$

A typical example are physical simulations, such as atmospheric and oceanographic simulations [GW08, p. 278]. Here, each state represents the model at a certain point in time, and each function F_i calculates the simulation of one time step. To prevent confusion of the simulated time with the computation time, we introduce the term *physical step* to always refer to the simulated time, not the computation time. More precisely, we say that F_i calculates (or simulates) the physical step i by mapping the previous state s_{i-1} to the state s_i.

$$s_0 \xrightarrow{F_1} s_1 \xrightarrow{F_2} \cdots \xrightarrow{F_\ell} s_\ell$$

Their composition F then describes the whole model:

$$F = F_\ell \circ \cdots \circ F_1$$
$$s_0 \xrightarrow{F} s_\ell$$

Written like that, evolutionary systems are a special case of mathematical models, but also describe the general case.

1. They are a *special case* in the sense that each state s_i depends only on s_{i-1}, while we would normally expect it to depend on all preceding states $s_0, s_1, \ldots, s_{i-1}$.
2. On the other hand, we could always define every state s_i to contain all intermediate states calculated so far, in which case *every calculation* could be represented as an evolutionary system.

To emphasize that we mean the first kind of models and not the second one, we demand that all states s_i have roughly the same size. More precisely, we demand that all s_i can be represented within the same amount of computer memory $\text{mem}(s_i)$, and that we have working implementations of all F_i which operate on those representions.

$$\text{mem}(s_0) = \text{mem}(s_1) = \cdots = \text{mem}(s_\ell)$$

In this thesis, we are concerned with the following question: If we are given an evolutionary system with ℓ physical steps and the initial state s_0, how do we compute the states in reverse order?

$$\left(s_0 \xrightarrow{?}\right) s_\ell \xrightarrow{?} s_{\ell-1} \xrightarrow{?} \ldots \xrightarrow{?} s_1 \xrightarrow{?} s_0$$

This question arises in several situations. For example, the evolutionary system may be a physical simulation displayed to the user, and once the user has seen all states $s_0, \ldots, s_\ell$, they want to see the simulation in reverse.

Another application is *algorithmic differentiation*, which is a technique to calculate derivatives, with applications in nonlinear optimization, solving partial differential equations and others. In particular, we are interested in the *reverse mode* of algorithmic differentiation, which calculates adjoint derivatives such as gradients. The reverse mode is not restricted to evolutionary systems, but those allow for a more memory-efficient treatment.[1] It requires, however, that the reverse mode has access to all states s_i in reverse order.

So how exactly do we perform a reverse computation of an evolutionary system? One simple approach is:

$$\left(s_0 \xrightarrow{F_\ell \circ \cdots \circ F_1}\right) s_\ell \xrightarrow{F_\ell^{-1}} s_{\ell-1} \xrightarrow{F_{\ell-1}^{-1}} \ldots \xrightarrow{F_2^{-1}} s_1 \xrightarrow{F_1^{-1}} s_0$$

That is, we first calculate s_ℓ from s_0 by applying $F_1, \ldots, F_\ell$. Then, one by one, we apply $F_\ell^{-1}, \ldots, F_1^{-1}$, which yield the states in reverse order. However, this is only possible if all functions F_i are invertible and we have working implementations of all F_i^{-1}. Therefore, this approach is not feasible for most models. The exact reasons vary from case to case, but are often a combination of:

1. The F_i are not injective and it is hard or impossible to choose the right preimage $s_{i-1} \in F_i^{-1}(s_i)$ when a state s_i is given.[2]
2. Computing all F_i^{-1} is orders of magnitude more expensive than computing all F_i.
3. Inversion of the F_i is mathematically possible, but numerically ill-conditioned. This happens for heat equations and other diffuse processes.

For those cases, where we do not have working implementations of the F_i^{-1}, reversal schedules come into play.

---

[1] This approach has been used successfully, for example, in seismic research. [Sym07]

[2] In some cases we can deal with this by storing additional hints in the result state, but in general this results in accumulating hints from previous states, making the states $s_0, s_1, s_2, \ldots$ growing from step to step, which is not what we want.

1.2. Parallel Reversal Schedules

Suppose we want to reverse an evolutionary system of ℓ physical steps

$$s_0 \xrightarrow{F_1} \cdots \xrightarrow{F_\ell} s_\ell$$

where we do not have working implementations of the F_i^{-1}. How do we achieve this?

One possibility is to simply save all states s_i as they are computed. This *total recall* strategy is depicted in Figure 1. The x-axis indicates the computation time, while the y-axis indicates the states s_i. Note that throughout this thesis, we assume that all F_i require roughly the same computation time, and define *one time unit* to be exactly that time.[3]

$$\text{time}(F_1) = \text{time}(F_2) = \cdots = \text{time}(F_\ell) = 1$$

The dots denote which states are available in computer memory at what point in time. The lines which connect the dots denote various actions.

The simplest action is *Checkpoint holding* (C), denoted by a solid horizontal line ($\hookleftarrow\!\!\!\rightarrow$), which means that the same state s_i is still available after one time unit has passed. This is often implemented by checkpointing, that is, copying the current computing state to a separate memory location. That checkpoint is then held in memory until it is restored at a later point in time.

Another action is *forward* (F), denoted by a solid diagonal line ($\nearrow$), which means that F_{i+1} is being applied to state s_i, so after one time unit state s_{i+1} is available.

This would be all we need. But for algorithmic differentiation, two more actions are needed and scheduled explicitly. These are depicted as dashed lines in Figure 1. To understand those actions, it is necessary to know that the reverse mode does not only need the states s_i in reverse order. It needs *all intermediate results* that appear during evaluation of all physical steps F_i, in reverse order. We meet this requirement by replacing the last F action with a *prepare* action (P), depicted as a dashed forward

---

[3]Note that [Wal99] also considered varying time(F_i), but only for sequential schedules, not for parallel schedules.
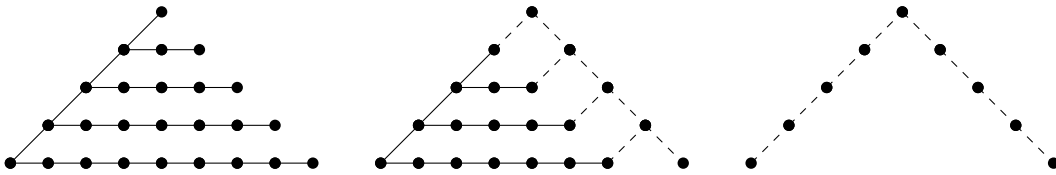

Figure 1.: Total recall schedules for the reversal of $\ell = 4$ physical steps. The middle schedule incorporate the *prepare* and *reverse* actions explicitly, as needed for algorithmic differentiation. The right schedule shows the plain reverse mode that contains only *prepare* and *reverse* actions without any checkpointing.

diagonal line (⋰). This also evaluates F_{i+1}, but stores all intermediate values into a separate stack in memory. This is also denoted as $\hat{F}_{i+1}$.

Every P action is followed by a *reverse* action (R), depicted as a dashed backward diagonal line (⋱). This starts or continues the actual reversal, retrieving all intermediate values in reverse order from the prepared stack. This is also denoted as $\bar{F}_{i+1}$. Throughout this thesis we assume that P and R actions have roughly the same computation complexity as F actions, that is:[4]

$$\text{time}(\hat{F}_i) = \text{time}(\bar{F}_i) = \text{time}(F_i) = 1$$

Figure 1 also demonstrates that during every R action, there is also the P action running for the previous physical step, except for the last R action. So for $\ell > 1$ this schedule requires at least two processes running in parallel. That is, it requires at least either a two-core processor or a cluster of two computers. Moreover, up to three checkpoints must be held in memory simultaneously. The total number of C actions is $\ell(\ell - 1)$, the total number of F actions is $\ell - 1$.

Note that this schedule runs in optimal computation time. Under the given preconditions, we can't do any better than an uninterrupted forward calculation followed by an uninterrupted reverse calculation. In this thesis we consider only that kind of schedules, also called *feasible parallel reversal schedules* [Wal99, p. 58], which means that we have always $t_M = 2\ell$, where t_M is the total computation time according to the schedule.[5]

While the *total recall* strategy is wasteful with memory, it is still a major improvement over the plain reverse mode with regard to memory usage. As shown in Figure 1, the plain reverse mode would create a huge stack of all intermediate values during the evaluation of all physical steps $F_1, \ldots, F_\ell$. This is a lot more than the memory required to store just all states $s_0, \ldots, s_\ell$.

But why should care about memory at all? The obvious reason is of course economics. There is a cost limit in attaching more RAM and hard drives to a computer system. The less obvious reason is speed, because the reverse mode is memory bound. Going from L3 cache to RAM already incurs a considerable penalty, even more so when going from RAM to solid-state drives and from there to hard drives. Since the memory is accessed sequentially, the access pattern is already optimal. The various caching systems won't provide much speedup here, as their main task is to transform random access patterns into sequential access patterns. In the end, the slowest memory in the chain will be the bottleneck. If "wasting" lots of CPU cycles through recalculation leads to avoiding the slow memory, this is a good tradeoff, as we are using processing resources that would otherwise waste their time waiting for memory components to respond. It was shown that this speedup is measurable not just on parallel systems, but also on a single computer with a single-core CPU. [GW08, pp. 293–294]

The oppsite of the *total recall* strategy is the *total recalculation* strategy, which is

---

[4]Note that [Wal99] also considered scenarios where the time of the P and R actions are whole-number multiples of the time of F actions.

[5]The exact formula in [Wal99, p. 58] is $t_M = (\ell - 1) + \hat{t} + \ell\bar{t}$, but our previous assumptions translate to $t = \hat{t} = \bar{t} = 1$, so this formula simplifies to $t_M = 2\ell$.

depicted in Figure 2. Here, every state s_i is recalculated from the base s_0. Apart from the obligatory P and R actions, there are only F actions in this schedule. C actions are only used to keep s_0 available.[6] In this schedule, up to tree processes run in parallel, while at most one checkpoint needs to be held in memory. The total number of C actions is $2\ell - 2$, the total number of F actions is $\frac{\ell(\ell-1)}{2}$.

Of course, there is a large degree of freedom between *total recall* and *total recalculation* schedules.

1.3. Relaxing Processor-Checkpoint Convertibility

In this thesis we are concerned with the following questions: Depending on the number of processes that can truly run in parallel on our hardware, what are the best feasible parallel reversal schedules? How many physical steps can be reversed in optimal time within what amount of memory?

So we have a discrete optimization problem with three parameters:

ℓ – number of physical steps to be reversed
p – number of processes that can run in parallel
k – number of states that fit into memory, to be used by processes and checkpoints

So at any point in time within a schedule, the number of actions (C, F, P, R) must be $\leq k$. Moreover, the number of F, P and R actions must be $\leq p$.

Since it doesn't make sense to start processes for whose execution we have no memory, we demand that:

$$p \leq k$$

Note that k has the same meaning as what it called *resources* in [Wal99], while p is an additional degree of freedom not present in [Wal99]. Due to the introduction of p, we relax the assumption of *processor-checkpoint convertibility* [Wal99, p. 62] which says that processors and checkpoints are interchangeable resources.

---

[6]It may be debatable whether keeping the initial state s_0 is *checkpointing* in the classic sense. However, in our systematics we consider it to be a C action, as it holds one state in memory for one time unit.
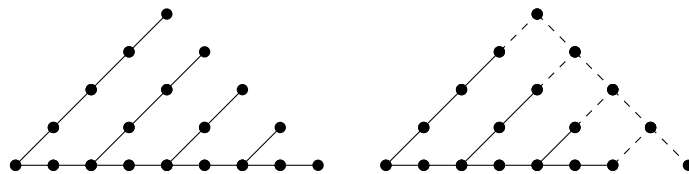

Figure 2.: Total recalculation schedules for the reversal of $\ell = 4$ physical steps. The right schedule incorporates the *prepare* and *reverse* actions explicitly, as needed for algorithmic differentiation.

This was a reasonable assumption in 1999, but today's latops have 8 GiB RAM shared by 4 cores. Assuming we have an evolutionary system with large states s_i, where each state consists of 10 million double precision floating point values ($\approx$80 MiB), we would have $k > 100$ but only $p = 4$. That ratio becomes worse when looking today's expensive computers, which have 1 TiB RAM shared by 120 cores, leading to $k > 13000$ versus $p = 120$. For evolutionary systems with smaller states, the ratio is even worse.

So we relax the processor-checkpoint convertibility and assume that processors (F, P, R actions) can be converted to checkpoints (C actions) by "doing nothing", but not vice versa. From that point of view, our additional parameter p describes how many of the k resources can be used as processors, while the remaining $k - p$ resources can only be used as checkpoints.

At a first glance, this seems to imply that the schedules developed in [Wal99] are only applicable if we choose p to be $p = k$. However, according to [Wal99, p. 79] the optimal schedules for k resources require just $p = \left\lceil \frac{k+1}{2} \right\rceil$ processes. So these schedules are applicable and optimal for all:

$$p \geq \left\lceil \frac{k+1}{2} \right\rceil$$

Nevertheless, for $p < \left\lceil \frac{k+1}{2} \right\rceil$ we need to develop new schedules, as the schedules of [Wal99] are not applicable. This will result in new schedules that require fewer processors by using more checkpoints and thus more memory.

1.4. Scope and Structure of this Thesis

While there has been interest in reversal schedules since at least 1973 [Ben73], the first systematic analysis of optimal schedules was published 1999 in the dissertation [Wal99]. Research continued for about 5 years, leading to another dissertation [Leh03] and various papers [GW00] [LW02] [Wal04], all within the same research group. Alas, this topic has never been picked up by other research groups. Today, books which cover reversal schedules all refer to the same results from a decade ago. [GW08, pp. 278–297] [NS12, pp. 142–147]

This diploma thesis is an attempt to continue the research on reversal schedules by relaxing the *processor-checkpoint convertibility*, which is a central assumption of [Wal99]. This introduces two challenges:

1. The central *binary decomposition* theorem [Wal99, p. 66] does not hold anymore.
2. The discrete optimization problem has three instead of two parameters: p, s, ℓ instead of s, ℓ.

To cope with those challenges, this thesis proposes and follows a more symbolic approach to reversal schedules. Also, to keep things manageable, this thesis focuses on parallel reversal schedules in the case $t = \hat{t} = \bar{t} = 1$.

It is structured as follows. In Chapter 2, a small formalism is developed for describing schedules, showing which properties still hold and which properties don't hold anymore

due to the relaxation. In Chapter 3, a comprehensive algebra for describing and analyzing the profiles of schedules is developed. Finally, this is used in Chapter 4 to build new optimal and suboptimal schedules, which are then compared to the already known schedules, but also to the result of an exhaustive search over small schedules.

1.5. Notations and Conventions

Throughout this thesis, we use the following notations and apply the following conventions:

$\mathbb{B}$ Set of Booleans, $\mathbb{B} = \{\text{true}, \text{false}\}$

2^A Power set of the set A

$f^{-1}(A)$ Preimage of the set A under the function f

χ_A Indicator function $\chi_A \colon \mathbb{R} \to \{0, 1\}$ of the subset $A \subseteq \mathbb{R}$

e_i The ith unit vector in $\mathbb{R}^n$

F_n The nth Fibonacci number with $F_1 = 1$, $F_2 = 1$ and $F_n = F_{n-1} + F_{n-2}$

$\mathbb{F}_n$ The finite field with n elements, where n is the power of a prime

A^* The free monoid over the set A

λ The empty word $\lambda \in A^*$

$\mathbb{Z}\langle A \rangle$ The free, associative, unitary $\mathbb{Z}$-algebra over the set A

Axis order   We follow the usual convention in mathematics that the time axis is the x-axis and points right, while the y-axis points upwards. So we deviate from [Wal99, p. 5], where the y-axis points downwards. We also deviate from [Leh03, p. 24], where the time axis is the y-axis and points downwards.

2. Algebra on Schedules

In this chapter we explain parallel reversal schedules and their operations in a formal way, assuming the relaxed processor-checkpoint convertibility. We will then check which central statements of [Wal99] still hold and which don't.

There are many ways to represent parallel reversal schedules formally, each with its own strengths and weaknesses in different areas. The algebraic representation presented here has been developed with the following goals: The definition of feasible parallel reversal schedules, as well as their composition, should be straight forward. Moreover, checkpoint persistence and processor persistence should be explainable within the formalism as clearly as within the visual representation.

The weakness of the algebra is that it is harder to see which exact actions are happening at a certain point in time. For reference, Appendix B shows how this can be done within the schedule algebra, but there is a more elegant way: We will simply introduce a separate algebra on schedule profiles, which will be the topic of Chapter 3, combining the best of both worlds.

2.1. Generic Parallel Schedules

In this section we will develop a generic algebraic representation of parallel schedules that works with an arbitrary set A of actions. However, to keep the examples closer to the problem at hand, we will stay within our set of actions as introduced in Section 1.2. We will explain how this algebra represents our schedules, and visualize the various operations. Finally, we will point out the corner cases in which the algebra differs from the visual language.

However, to keep the examples closer to the problem at hand, we will stay within our set of actions as introduced in Section 1.2.

Definition 1 (Actions)**.** The set of actions A is:

$$A = \{C, F, P, R\} \tag{1}$$

where C, F, P and R are to be interpreted as the actions introduced in Section 1.2.

Before getting to schedules, we define an intermediate structure, a *task*, by which we mean a sequential chain of actions that are to be executed one after another.[1]

---

[1]We could also call this a *sequential schedule* as opposed to a *parallel schedule*, but we choose the term *task* to prevent confusion. Throughout this thesis, *schedule* always means *parallel schedule*.

Definition 2 (Task). A *task* w is a finite sequence, or word, of actions. That is, the set of tasks is the free monoid over A:

$$w \in A^*$$

The empty task is denoted as the empty word $\lambda \in A^*$.

Definition 3 (Task duration). The *duration* $|w|$ of a task $w \in A^*$ is the number of its actions:

$$|w| = \begin{cases} 0 & \text{for } w = \lambda \\ n & \text{for } w = a_1 \cdots a_n, a_i \in A \end{cases} \tag{2}$$

This is also known as the *length of a word*, the *length of a monomial* [Coh85, p. 60] or the *degree of a monomial*.

Remark. Here we assume that every action needs exactly one time unit to run. We can so without loss of generality, because if the actions had different rational durations, we could redefine our time unit according to the lowest common denominator and split each action into repeated smaller actions. For example, if the P action took twice as long as the F action, we could introduce a new action P' and redefine P to be the task $P'P'$.

Definition 4 (Number of action occurrences). For every action $a \in A$ and task $w \in A^*$ we define $c_a(w)$ to be the number of occurrences of a in w:
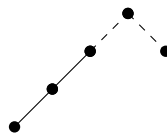
$$c_a(w) = \begin{cases} 0 & \text{for } w = \lambda \\ |\{i \in \{1, \ldots, n\} \mid b_i = a\}| & \text{for } w = b_1 \cdots b_n, b_i \in A \end{cases} \tag{3}$$

**Lemma 5** (Task duration inequality). *For all $a \in A, w \in A^*$ we have:*

$$|w| \geq c_a(w) \tag{4}$$

Proof. $|w|$ counts all actions of w while $c_a(w)$ counts only some of them. $\qquad \square$

Example 6. The task $w = FFPR$, where $c_F(w) = 2$, means two forward actions followed by a prepare and a reverse action. It describes the direct way to reverse the third physical step:

Example 7. The task $w = FCCPR$, where $c_F(w) = 1$, describes the reversal of the second physical step, with a delay in between:
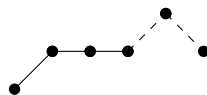
Example 8. The task $CCCCPR$, where $c_F(w) = 0$, describes the reversal of the first physical step, with a long delay at the beginning:
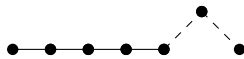
Motivation. Now we want to define a "+" operation to overlay multiple tasks to a schedule. Here it must be possible to unify the front overlaps, which are the common prefixes of the tasks. To achieve this, we choose a simple noncommutative algebra over A^* that provides distributivity.

Definition 9 (Schedule). A *schedule f over A* is an element of the free, associative, unitary $\mathbb{Z}$-algebra [Coh85, p. 59] over A:

$$f \in \mathbb{Z}\langle A \rangle$$

Definition 10 (Schedule duration). The *duration $|f|$* of a schedule $f \in \mathbb{Z}\langle A \rangle$ is the duration of its longest task:

$$|f| = \begin{cases} 0 & \text{for } f = 0 \\ \max\{|w_1|, \ldots, |w_n|\} & \text{for } f = d_1 w_1 + \cdots + d_n w_n, w_i \in A^*, d_i \in \mathbb{Z} \setminus \{0\} \end{cases}$$

In the context of algebra, this is also known as the *degree*. [Coh85, p. 60] In the context of reversal schedules, this is also known as t_M. [Wal99, p. 58]

Definition 11 (Number of tasks). For every schedule $f \in \mathbb{Z}\langle A \rangle$ we define $\ell(f)$ to be the number of tasks:

$$\ell(f) = \begin{cases} 0 & \text{for } f = 0 \\ d_1 + \cdots + d_n & \text{for } f = d_1 w_1 + \cdots + d_n w_n, w_i \in A^*, d_i \in \mathbb{Z} \setminus \{0\} \end{cases} \tag{5}$$

Remark. $\mathbb{Z}\langle A \rangle$ is the noncommutative analogue to the ring $\mathbb{Z}[A]$ of polynomials. Although the only coefficients of interest are 1 and 0, we choose the base ring to be $\mathbb{Z}$, because $\mathbb{Z}$ does not impose an additional structure on the algebra.[2] A term like $2 \cdot FFF$ is then to be interpreted as executing the task FFF twice in parallel, wasting an additional processor for no use. A negative term like $-1 \cdot FFFPR$ may occur when considering the difference of two schedules $f - g$, meaning that f is identical to g except that g contains an additional task $FFFPR$.

Remark. We use the distributivity in $\mathbb{Z}\langle A \rangle$ to extract common prefixes of tasks within a schedule. The reason, of course, is that starting from the same state, equal tasks (or task prefixes) will produce equal results at the same points in time, so we will calculate them only once. After factoring out prefixes to the left as much as possible, the term describes our schedule. More precisely, the abstract syntax tree of a maximally left-factored term

---

[2]For example, if we had chosen $\mathbb{F}_2$ to be our base ring, we would have exactly the coefficients 1 and 0, but adding a task to a schedule which already contains that task would make it disappear: For $S = FPR + CCPR$ we would have $S + CCPR = FPR$, which is not what we want.

has the same tree structure as the schedules, when ignoring the connections between successive reversals and collapsing all intermediate expressions $a \cdot \square$ and $a \cdot (\square + \square)$ with $a \in A$ into single nodes.

Example 12. Adding the tasks of Examples 6 to 8 results in the following schedule:

$$f = FFPR + FCCPR + CCCCPR$$

It describes the reversal of three physical steps via the total recall strategy:
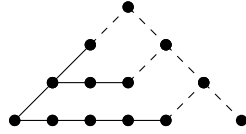
When ignoring the connections between successive reversals, this graph forms a tree, the schedule tree. Now we factor out all prefixes to the left, which in this case is just the F prefix of the first two tasks:
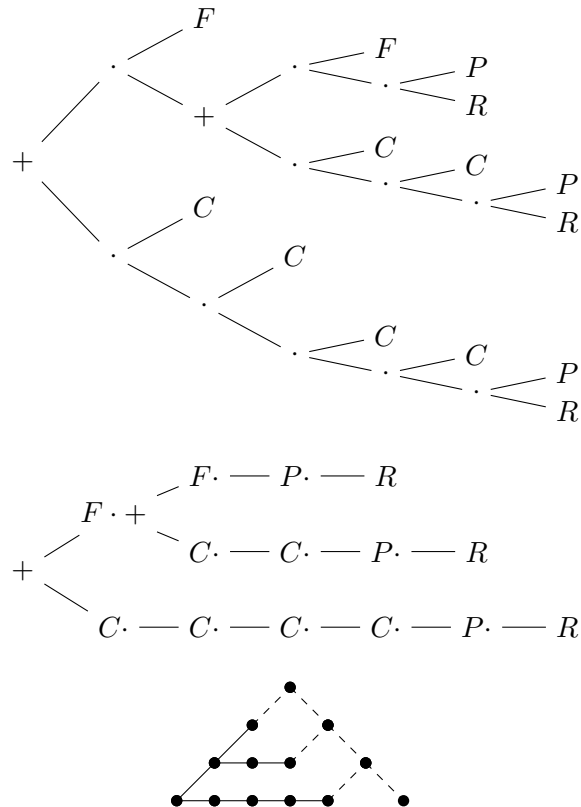
$$f = F(FPR + CCPR) + CCCCPR$$

Figure 3.: AST, collapsed AST and schedule tree of $F(FPR + CCPR) + CCCCPR$.

The collapsed abstract syntax tree has the same structure as the schedule tree, which is demonstrated in Figure 3.

Remark. Factoring out to the right will not result in the subtasks to be unified. It is significant to which side we are factoring out.

Example 13. In the same schedule as before we factor out all PR suffixes to the right:

$$f = FFPR + FCCPR + CCCCPR$$
$$= (FF + FCC + CCCC) \cdot PR$$

These PR subtasks all start on different states, and even more so, at different times. So it makes no sense to unify them, we must repeat them:
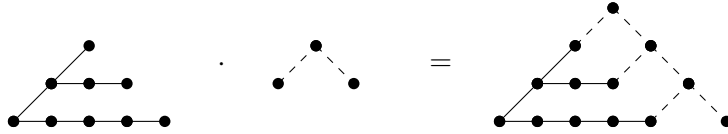
Remark. More generally, in our algebra we deliberately unify *only common prefixes* and nothing else. This is an important difference between our algebra and the visual schedule language. We don't even unify tasks that produce the same state at the same point in time, if they have different prefixes.

Although this way the algebra introduces a "punishment" on some schedules, this happens only to schedules that are inefficient anyway, so it makes no sense to introduce that additional complexity into our algebra as this will not provide any benefit in return.

More precisely, let $u, v \in A^*$ be two tasks without a common prefix that produce the same state after the same amount of time. Let $w \in A^*$ be an arbitrary common continuation of both, and $g, h \in \mathbb{Z}\langle A \rangle$ be their diverging ends. Then the schedule $f = uwg + vwh$ consists of two entirely separate calculations. The modified schedule $f' = uwg + uwh = uw(g + h)$ is always more efficient, even if we would unify the two common sub tasks w in f.

Example 14. The overlap in the following schedule:

$$f = FC\underline{CF}C + CF\underline{CF}F$$

is not unified:
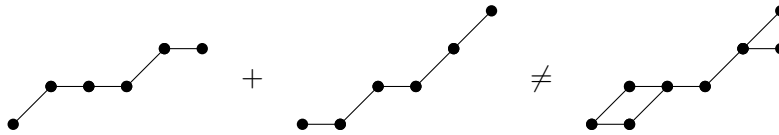
In other words, the schedule f is not equivalent to f':

$$f \neq f' = FC + CF\underline{CF}(F + C)$$

where f' is the schedule at the right-hand side of the visual equation above.

2.2. Parallel Reversal Schedules

So far, our algebra is very generic and could have been defined for any set of actions A. Now we introduce the specifics of reversal schedules.

Definition 15 (Parallel reversal schedule). A schedule $f \in \mathbb{Z}\langle A \rangle$ is a *parallel reversal schedule* (PRS) if there exist $\ell, d \in \mathbb{Z}_{\geq 1}$ and $w_1, \ldots, w_\ell \in \{C, F\}^*$ such that:

$$f = w_1 PR + \cdots + w_\ell PR \tag{6}$$

where:

$$c_F(w_i) = i - 1 \qquad i = 1, \ldots, \ell \tag{7}$$
$$|w_i| = d - 1 - i \qquad i = 1, \ldots, \ell \tag{8}$$

Remark. This definition implies $\ell(f) = \ell$ and $|f| = |w_1 PR| = |w_1| + 2 = d$.

Remark. The equations (6) to (8) are to be interpreted as follows:

(6) All tasks consist of a sequence of C and F actions, followed by a PR. This rules out nonsense tasks such as $CPCRCF$:

(7) The tasks $w_1 PR, \ldots, w_\ell PR$ each reverse the first, second, …, ℓth physical step. This rules out schedules whose reversals are not vertically connected, such as $CCCPR + FFPR$:
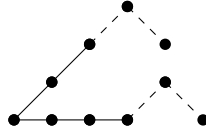
(8) All reversals line up diagonally towards the end. This rules out schedules whose reversals are not horizontally connected, such as $CCCPR + FPR$:
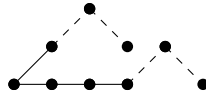
Definition 16 (Reach). The *reach* of a PRS f is the number of reversed physical steps, which by (7) is also the the number of tasks $\ell(f)$.

Example 17. The smallest PRS is:

$$f = PR$$

$$\ell(f) = 1$$
$$|f| = 2$$

Example 18. The second smallest parallel reversal schedule is:

$$f = CCPR + FPR$$
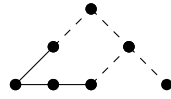
$$\ell(f) = 2$$
$$|f| = 4$$

Example 19. A more complex parallel reversal schedule is:

$$
\begin{aligned}
f = \quad & CCCCCCCCCPR \\
+ & CCCCFCCCCPR \\
+ & CCCCFCCFPR \\
+ & FCFCCFCPR \\
+ & FFFFCCPR \\
+ & FFFFFPR
\end{aligned}
$$
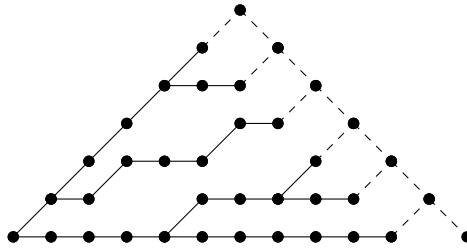
$$\ell(f) = 6$$
$$|f| = 12$$

Motivation. Definition 15 does not demand a minimal computation time. It includes schedules such as $f = CCCPR + FCPR$ with $\ell(f) = 2$ and $|f| = 5 > 4$:
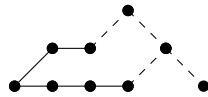
These will be ruled out by the *feasible* parallel reversal schedules.

Lemma 20 (Minimal reversal duration)**.** *The minimal duration of a parallel reversal schedule f is twice its reach. That is,*

$$|f| \geq 2\ell(f)$$

and this inequality is sharp.

Proof. Let $d, w_1, \ldots, w_\ell$ be as in Definition 15:

$$f = \sum_{i=1}^{\ell} w_1 PR + \cdots + w_\ell PR$$

The inequality follows from Lemma 5 for w_ℓ and applying (7) and (8):

$$|f| = d = |w_\ell| + 1 + \ell \geq c_F(w_\ell) + 1 + \ell = 2\ell = 2\ell(f)$$

Equality holds whenever $|w_\ell| = c_F(w_\ell)$, that is, whenever $w_\ell = F^\ell$. Among others, this is the case for all schedules that implement the *total recall* strategy:

$$f = \sum_{i=1}^{\ell} F^{i-1} C^{2\ell-2i} PR$$

So this inequality is sharp. $\square$

Definition 21 (Feasible parallel reversal schedule). A PRS f is *feasible* (FPRS) if it has minimal duration, that is:

$$|f| = 2\ell(f)$$

In other words, there exist $\ell \in \mathbb{Z}_{\geq 1}$ and $w_1, \ldots, w_\ell \in \{C, F\}^*$ such that:

$$f = \sum_{i=1}^{\ell} w_1 PR + \cdots + w_\ell PR$$
$$c_F(w_i) = i - 1$$
$$|w_i| = 2\ell - 1 - i$$

Remark. Definition 21 corresponds to the feasible parallel reversal schedules as introduced in [Wal99, p.58] in the case $\hat{t} = \bar{t} = 1$.

Definition 22 (Resources profile). The *resources profile* of a schedule f is a step function (piecewise constant function) with support $[0, |f|)$:

$$\operatorname{resp} f \colon \mathbb{R} \to \mathbb{R}$$

where $(\operatorname{resp} f)(t)$ is the number of actions at any point in time $t \in \mathbb{R}$.

Definition 23 (Processes profile). The *processes profile* of a schedule f is a step function (piecewise constant function) with support $[0, |f|)$:

$$\operatorname{procp} f \colon \mathbb{R} \to \mathbb{R}$$

where $(\operatorname{procp} f)(t)$ is the number of F, P and R actions at any point in time $t \in \mathbb{R}$.

Definition 24 (Usage)**.** The processes and resources usage of a schedule f is the respective resource peak over time:

$$\text{res}\, f = \max(\text{resp}\, f)$$
$$\text{proc}\, f = \max(\text{procp}\, f)$$

Remark. Since we don't have negative times and our actions cover discrete time durations, processes and resources, we could have defined $\text{resp}\, f$ and $\text{procp}\, f$ to be nonnegative integer functions $\mathbb{Z}_{\geq 0} \to \mathbb{Z}_{\geq 0}$. However, prefer them to be functions $\mathbb{R} \to \mathbb{R}$ as this fits better into the algebra on profiles we will establish in Chapter 3.

Definition 25 (Optimization problem, reach maximal)**.** Given $r, p \in \mathbb{Z}_{\geq 0}$, find a FPRS f with

$$\text{res}\, f \leq r$$
$$\text{proc}\, f \leq p$$

for which

$$\ell(f) \text{ is maximal}$$

Definition 26 (Optimization problem, resource minimal)**.** Given $\ell, p \in \mathbb{Z}_{\geq 0}$, find a FPRS f with

$$\ell(f) \leq \ell$$
$$\text{proc}\, f \leq p$$

for which

$$\text{res}\, f \text{ is minimal}$$

2.3. Decomposition and Composition

Under the assumption of processor-checkpoint convertibility, two important properties were proved by previous research, *checkpoint persistence* [Wal99, p. 64] and *processor persistence* [Wal99, p. 65]. Those led to the theorem of *binary decomposition* [Wal99, p. 66], which was the base of all further analysis.

With relaxed processor-checkpoint convertibility, this line of reasoning is no longer applicable. Although we can show that checkpoint persistence still holds, there is no evidence that processor persistence holds. Finally, we will describe its counterpart, the *composition* [Wal99, p. 74], which is still applicable.

Definition 27 (Checkpoint-persistent schedule). A schedule f is *checkpoint-persistent* if its maximally left-factored representation contains only the following subtasks, where $n \in \mathbb{Z}_{\geq 1}, w \in A^*$:

$$C^n$$
$$C^n PR$$
$$Fw$$
$$PR$$

In other words: Once a subtask starts with a C action, it will continue with C actions until the next fork.

Example 28. The following schedule f is not checkpoint-persistent. The violating subtask is underlined and marked in the visual schedule.

$$f = F(F^3(FPR + C^2PR) + \underline{C^3FC}(FPR + C^2PR)) + C^8(FPR + C^2PR)$$
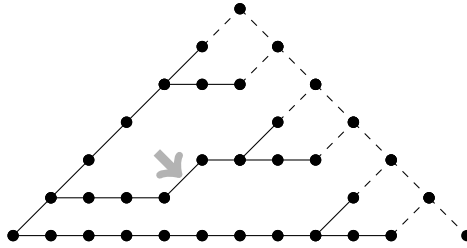

Definition 29 (Processor-persistent schedule). A schedule f is *processor-persistent* if its maximally left-factored representation contains only the following tasks, where $n \in \mathbb{Z}_{\geq 1}, w \in A^*$:

$$F^n$$
$$F^n PR$$
$$Cw$$
$$PR$$

In other words: Once a subtask starts with an F action, it will continue with F actions until the next fork.

Example 30. The following schedule f is not checkpoint-persistent. The violating subtask is underlined and marked in the visual schedule.

$$f = F^4(FPR + C^2PR) + C^3(\underline{F^2CFPR} + C^3(F(FPR + C^2PR) + C^4PR))$$
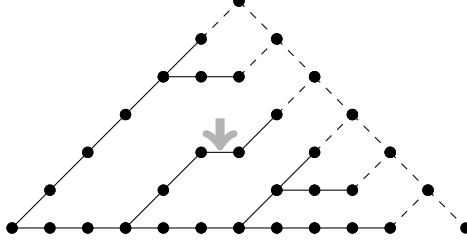
Theorem 31 (Checkpoint persistence)**.** *Let f be a FPRS of reach ℓ that uses p processes and k resources. Then there exists a checkpoint-persistent FPRS f' of the same reach ℓ. f' uses no more than p processes and k resources.*

Corollary 32. *In the search for optimal FPRS it is sufficient to restrict the search to checkpoint-persistent schedules.*

Proof. This proof implements the same idea as in [Wal99, p. 64], but is expressed through the notions of our schedule algebra.

If f is checkpoint-persistent, we set $f' = f$. Otherwise, f ist not checkpoint-persistent. That is, within its maximally left-factored representation f contains a violating subtask:

$$C^n F \qquad \text{with } n \in \mathbb{Z}_{\geq 1}$$

Let $w \in A^*$ be the preceding task, let $h_1 \in \mathbb{Z}\langle A \rangle$ be the following subschedule, let $Fh_2 \in \mathbb{Z}\langle A \rangle$ be the possibly existing other fork after w, and $g \in \mathbb{Z}\langle A \rangle$ the remaining schedule, that is, the sum of all tasks in f that to not have w as prefix. (Note that it is possible that $w = \lambda$, $h_2 = 0$ and $g = 0$.) Then:

$$f = w(C^n F h_1 + F h_2) + g$$

Now we apply the following transformation of f, which fixes this particular violation:

$$w(C^n F h_1 + F h_2) + g \quad \mapsto \quad wF(C^n h_1 + h_2) + g$$

At time $|w|$, the new schedule starts one fewer C action. At time $|w| + n$, it starts one fewer F action and possibly one more C action. (It won't if h_2 starts with C^m.) At all other points in time, both schedules have identical usage. So the new schedule uses no more than p processes and k resources.

This transformation can be iterated until all violations of checkpoint-persistence are rectified. Within each iteration a C action moved to the right, but there is never any C action moved to the left. So this cannot continue infinitely.

Let f' be the final result after which no further iteration is possible. f' must be checkpoint-persistent, otherwise one more iteration would be possible. Moreover, f' uses no more than p processes and k resources.

It remains to show that f' is a FPRS. Representing f as of Definition 21, the transformation preserves $|w_i|$ and $c_F(w_i)$ for all tasks w_i, hence it also preserves the duration $|f|$. Since it doesn't add or remove any task, it preserves the reach ℓ. So the transformation preserves FPRS. Since f' results from iterated transformation, it is also a FPRS. □

Example 33. Figure 4 shows a schedule f that is not checkpoint-persistent and the corresponding checkpoint-persistent schedule f' after one iteration of the transformation described in the proof.

$$f = F(F^3(FPR + C^2PR) + \underline{C^4}\underline{F}(FPR + C^2PR)) + C^8(FPR + C^2PR)$$
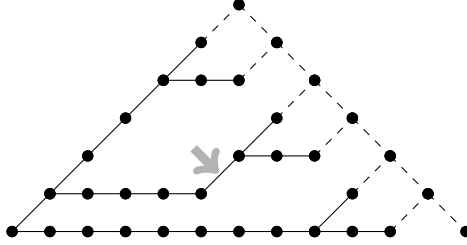
$$f' = F\underline{F}(F^2(FPR + C^2PR) + \underline{C^4}(FPR + C^2PR)) + C^8(FPR + C^2PR)$$
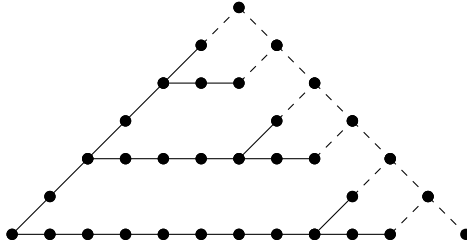
Figure 4.: Establishing checkpoint persistence.

Remark. The analogue transformation for processor persistence is:

$$w(F^nCh_1 + Ch_2) + g \quad \mapsto \quad wC(F^nh_1 + h_2) + g$$

At time $|w|$, the new schedule starts one fewer F action. At time $|w| + n$, it starts one fewer C action and possibly one more F action. (It won't if h_2 starts with F^n.) If the original schedule already used p processes at time $|w| + n$, and h_2 does not start with F^n, the new schedule will use $p + 1$ processes.

This means that we cannot apply this transformation in general, which is why the proof of [Wal99, p. 65] is not applicable here.

Example 34. Figure 5 demonstrates the worst case. It shows a schedule f that is not processor-persistent and the corresponding processor-persistent schedule f' after four iterations of the transformation. f' uses more processes than f.

$$f = F^4 PR + C(\underline{F^2 C}(FPR + C^2 PR) + C^2(\underline{FC^3}PR + C^5 PR))$$
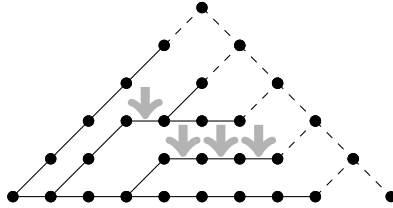
$$f' = F^4 PR + C\underline{C}(\underline{F^2}(FPR + C^2 PR) + C\underline{C^3}(\underline{F}PR + C^2 PR))$$
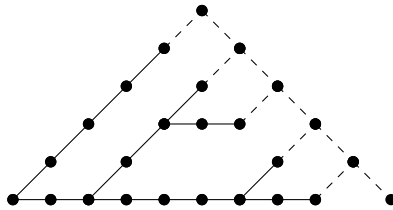
Figure 5.: Establishing processor persistence after four iterations. Schedule f uses $p = 2$ processes, while f' uses $p = 3$ processes.

Definition 35 (Binary-decomposable schedule)**.** A schedule f is a *binary-decomposable schedule* if it is checkpoint-persistent and processor-persistent.

That is, the maximally left-factored representation of f contains only the following tasks, where $n \in \mathbb{Z}_{\geq 1}$:

$$F^n$$
$$F^n PR$$
$$C^n$$
$$C^n PR$$
$$PR$$

Example 36. The schedules f' in Figures 4 and 5 are both binary-decomposable.

Remark. A binary-decomposable schedule is fully determined by its forks, which is a convenient property. However, due to the lack of processor-persistence, we cannot assume

that optimal FPRS are binary decomposable in general. Nevertheless, we can apply the inverse operation *binary composition* and will use that extensively to generate optimal and suboptimal schedules in Chapter 4.

Definition 37 (Primitive schedule)**.** The *elementary schedule* ε is:

$$\varepsilon = PR$$

Definition 38 (Schedule composition)**.** The *composition* (f, g) of two FPRS f and g is:

$$(f, g) = F^{\frac{|g|}{2}} f + C^{|f|} g \tag{9}$$

Remark. The composition (f, g) is a FPRS with reach $\ell((f, g)) = \ell(f) + \ell(g)$ and duration $|(f, g)| = |f| + |g|$.

Example 39. The composition of the following schedules:

$$f = FPR + C^2 PR \qquad\qquad \ell(f) = 2$$
$$g = F^2 PR + CFCPR + C^4 PR \qquad\qquad \ell(g) = 3$$

yields:

$$(f, g) = F^3(FPR + C^2 PR) + C^4(F^2 PR + CFCPR + C^4 PR)$$
$$= F^4 PR + F^3 C^2 PR + C^4 F^2 PR + C^5 FCPR + C^8 PR$$
$$\ell((f, g)) = 10$$
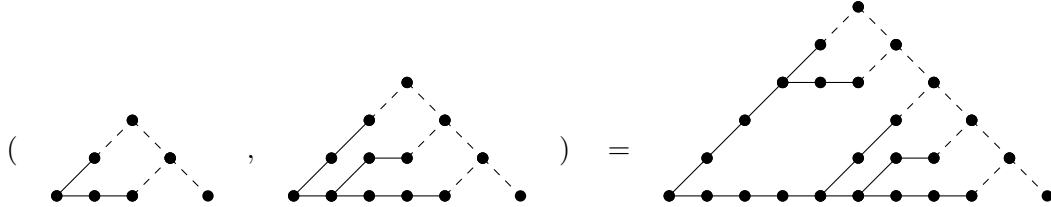
Proposition 40. *If f and g are binary-decomposable, their composition (f, g) is binary-decomposable, too.*

Proof. Let f and g be maximally factored to the left, then their subtasks are only the ones listed in Definition 35. The composition (f, g) add two more subtasks to this, which are both allowed by Definition 35. □

Example 41. All binary-decomposable schedules up to reach $\ell = 4$ are depicted in Figure 6.
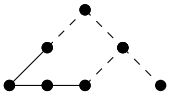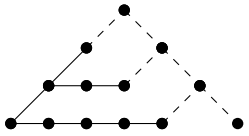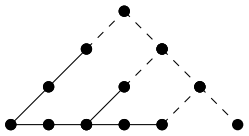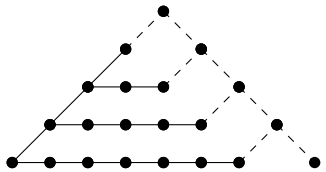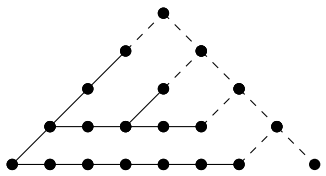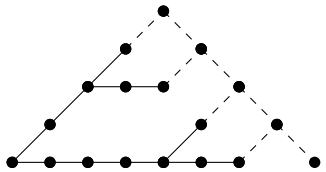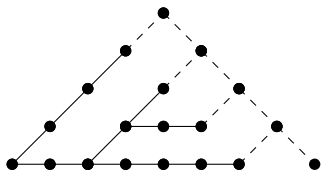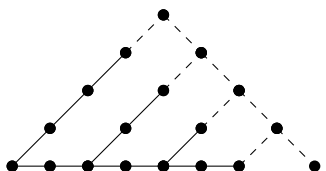
$$\cdot\;\cdot\;\cdot \quad = \quad \varepsilon \quad\quad\quad\quad = \quad PR$$

$$\cdot\;\cdot\;\cdot \quad = \quad (\varepsilon, \varepsilon) \quad\quad\quad = \quad FPR + C^2PR$$

$$\cdot\;\cdot\;\cdot \quad = \quad ((\varepsilon, \varepsilon), \varepsilon) \quad\quad = \quad F^2PR + FC^2PR + C^4PR$$

$$\cdot\;\cdot\;\cdot \quad = \quad (\varepsilon, (\varepsilon, \varepsilon)) \quad\quad = \quad F^2PR + C^2FPR + C^4PR$$

$$\cdot\;\cdot\;\cdot \quad = \quad (((\varepsilon, \varepsilon), \varepsilon), \varepsilon) \quad = \quad F^3\varepsilon + F^2C^2\varepsilon + FC^4\varepsilon + C^6\varepsilon$$

$$\cdot\;\cdot\;\cdot \quad = \quad ((\varepsilon, (\varepsilon, \varepsilon)), \varepsilon) \quad = \quad F^3\varepsilon + FC^2F\varepsilon + FC^4\varepsilon + C^6\varepsilon$$

$$\cdot\;\cdot\;\cdot \quad = \quad ((\varepsilon, \varepsilon), (\varepsilon, \varepsilon)) \quad = \quad F^3\varepsilon + F^2C^2\varepsilon + C^4F\varepsilon + C^6\varepsilon$$

$$\cdot\;\cdot\;\cdot \quad = \quad (\varepsilon, ((\varepsilon, \varepsilon), \varepsilon)) \quad = \quad F^3\varepsilon + C^2F^2\varepsilon + C^2FC^2\varepsilon + C^6\varepsilon$$

$$\cdot\;\cdot\;\cdot \quad = \quad (\varepsilon, (\varepsilon, (\varepsilon, \varepsilon))) \quad = \quad F^3\varepsilon + C^2F^2\varepsilon + C^4F\varepsilon + C^6\varepsilon$$

Figure 6.: Small schedule compositions up up to reach $\ell = 4$.

3. Algebra on Profiles

In this chapter we concentrate on the *profile* of a schedule, that is, the function that describes how many resources or processes are used at each point in time. Profiles are essentially step functions (piecewise constant functions), but we describe them in a slightly different way that will fit our purpose better.

Our main interest is to find a convenient description of how the profiles combine when two schedules are combined. Also, we will be able to describe exactly the difference between the two possibilities to combine three schedules. The main goal is, of course, to decribe upper limits of profiles. That is, we want to verify that our schedules do not exceed the number of processes and resources available.

While no formal language is ever perfect, the formalism developed here is at least powerful enough such that most lemmas can be proven solely by mechanically applying transformation rules (equations) of previous lemmas. Although much effort went into the proofs to make them concise, most could have been developed by simply expanding all terms as much as possible, then comparing the results. To provide a complete formalism, all possible operator relationships are studied, even if they do not contribute directly to a larger theorem. Some interesting connections between the algebra developed here and polynomials are described in Appendix C.

3.1. Profile Space

Definition 42 (Primitive functions)**.** For every $t \in \mathbb{R}_{\geq 0}$, the *primitive function* φ_t is:

$$\varphi_t \colon \mathbb{R} \to \mathbb{R}$$
$$\varphi_t = \chi_{[t,\infty)}$$

where χ_A denotes the indicator function as listed in Section 1.5. Further, we define $\mathcal{B}$ to be the set of primitive functions:

$$\mathcal{B} = \{\varphi_t \mid t \in \mathbb{R}_{\geq 0}\}$$

Definition 43 (Profile space)**.** Let $(\mathbb{R}^{\mathbb{R}}, +, \cdot, \leq)$ be the partially ordered $\mathbb{R}$-linear space[1] of all functions $\mathbb{R} \to \mathbb{R}$, where $+$, $\cdot$ and $\leq$ operate pointwise, as usual.

---

[1] A *partially ordered linear space* is a preordered linear space whose preorder $\leq$ is also a partial order. A *preordered linear space* is an F-linear space V with a compatible preorder $\leq$. That is, for all $x, y, z \in V, \lambda \in F$: $x \leq y \implies x + z \leq y + z$ and $x \geq 0, \lambda \geq 0 \implies \lambda \cdot x \geq 0$. For every set A, the $\mathbb{R}$-linear space $\mathbb{R}^A$ of functions $A \to \mathbb{R}$, equipped with the pointwise defined partial order $\leq$, forms a partially ordered linear space. [Bou87, p. II.12]

Within that space, we define the $\mathbb{R}$-linear subspace of *profiles* $\mathbb{P} \subset \mathbb{R}^{\mathbb{R}}$ to be the linear span of all primitive functions:

$$\mathbb{P} = \mathrm{span}_{\mathbb{R}}(\mathcal{B})$$

That is, for each *profile* $f \in \mathbb{P}$ there exist $t_1, \ldots, t_n \in \mathbb{R}_{\geq 0}$ and $a_1, \ldots, a_n \in \mathbb{R}$ such that:

$$f \colon \mathbb{R} \to \mathbb{R}$$
$$f = \sum_{i=1}^{n} a_i \varphi_{t_i}$$

Remark. For the problem at hand, it would have been sufficient to restrict profiles to integer arguments and values.[2] However, for the algebra introduced here that wouldn't simplify much. It would also complicate the usage of the arrangement operation as of Definition 59, where we divide by 2. So we will build this slightly more general framework, which is also capable of handling schedules with non-integer times as well as non-integer resource usage.

Remark. Note that although we don't consider negative time values, the profile functions are formally defined for those, too. Since we don't include any primitive functions φ_t with $t < 0$, all profile functions are zero for negative times. This may seem strange, but prevents many edge cases in the following definitions and proofs, especially Definition 49 and Lemma 50.

Lemma 44 (Profile values)**.** *For every* $f \in \mathbb{P}$, $f = \sum_{i=1}^{n} a_i \varphi_{t_i}$ *with* $t_1 < \cdots < t_n$, *the function values are:[3]*

$$t \in [t_k, t_{k+1}) \quad \implies \quad f(t) = \sum_{i=1}^{k} a_i$$

for all $t \in \mathbb{R}$ *and* $k \in \{0, \ldots, n\}$ *with* $t_0 = -\infty$ *and* $t_{n+1} = \infty$. *That is:[4]*

$$f(t) = \begin{cases} 0 & t \in (-\infty, t_1) \\ a_1 & t \in [t_1, t_2) \\ a_1 + a_2 & t \in [t_2, t_3) \\ \ldots \\ a_1 + a_2 + \cdots + a_{n-1} & t \in [t_{n-1}, t_n) \\ a_1 + a_2 + \cdots + a_{n-1} + a_n & t \in [t_n, \infty) \end{cases}$$

---

[2]We would then have $\varphi_t \colon \mathbb{Z} \to \mathbb{Z}$ with $t \in \mathbb{Z}_{\geq 0}$. Also, we would be dealing with free $\mathbb{Z}$-modules instead of linear spaces over $\mathbb{R}$.

[3]We follow the convention that the empty sum evaluates to zero: $\sum_{i=1}^{0} a_i = 0$.

[4]Note that the first condition $t \in (-\infty, t_1)$ is equivalent to $t \in [-\infty, t_1)$, because $t \in \mathbb{R}$.

Proof. First of all, note that t_0 and t_{n+1} are conveniently defined such that:

$$t_0 < t_1 < \cdots < t_n < t_{n+1}$$

Now, let $k \in \{0, \ldots, n\}$ and $t \in [t_k, t_{k+1})$. It follows that:

$$
\begin{aligned}
f(t) &= \sum_{i=1}^{n} a_i \cdot \varphi_{t_i}(t) \\
&= \sum_{i=1}^{n} a_i \cdot \chi_{[t_i, \infty)}(t) \\
&= \sum_{i=1}^{k} a_i \cdot \underbrace{\chi_{[t_i, \infty)}(t)}_{t \geq t_i} + \sum_{i=k+1}^{n} a_i \cdot \underbrace{\chi_{[t_i, \infty)}(t)}_{t < t_i} \\
&= \sum_{i=1}^{k} a_i \cdot 1 + \sum_{i=k+1}^{n} a_i \cdot 0 \\
&= \sum_{i=1}^{k} a_i \qquad \qquad \qquad \square
\end{aligned}
$$

Remark. Lemma 44 shows that profiles are step functions whose step intervals are closed on the left and open on the right.

However, our representation deviates from the classic representation in that our base intervals are not disjoint, but all overlapping towards $+\infty$. This causes an implicit summation of the coefficients, which is intentional and will simplify a lot.

For example, compare the following representations of the resources profile of schedule $((\varepsilon, \varepsilon), \varepsilon)$ as shown in Figure 7. The coefficients in the classic disjoint representation are the partial sums of the coefficients in our representation:

$$
\begin{aligned}
\text{resp}\,((\varepsilon, \varepsilon), \varepsilon) &= 2\varphi_0 + \varphi_2 - \varphi_4 - \varphi_5 - \varphi_6 && \text{(ours)} \\
&= 2\chi_{[0,2)} + 3\chi_{[2,4)} + 2\chi_{[4,5)} + \chi_{[5,6)} && \text{(classic)}
\end{aligned}
$$

Our representation enables us to think of the term "$a\varphi_t$" as an event that happens at time t and from then on will change resource usage by a. So φ_2 could describe the start of some computation at time 2 that consumes one resource unit. Then, $-\varphi_5$ could describe the end of that computation at time 5, freeing the one resource unit. Their sum, $\varphi_2 - \varphi_5$ would then describe a schedule in which exactly these two events happen, and nothing else. This is visualized in Figure 7.

Corollary 45 (Profile image). *For every $f \in \mathbb{P}$, the image of f is:*

$$
\{f(t) \mid t \in \mathbb{R}\} \quad = \quad \left\{ \sum_{i=1}^{k} a_i \;\middle|\; k = 0, \ldots, n \right\}
$$
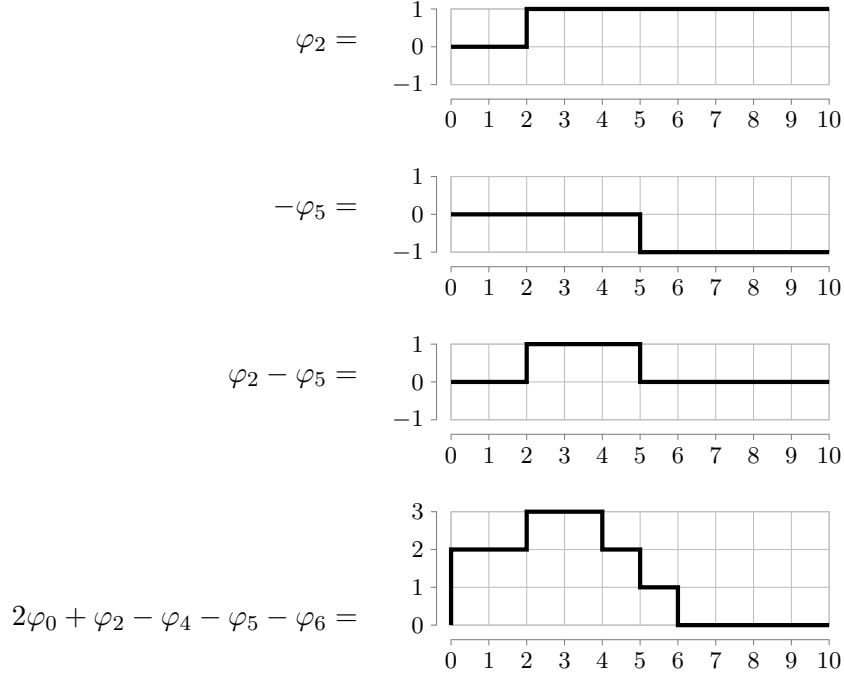
Figure 7.: Simple profile functions. The last diagram decribes the resources profile of schedule $(\varepsilon, (\varepsilon, \varepsilon))$ as depicted in Figure 6.

Lemma 46. *The primitive functions as of Definition 42*

$$\mathcal{B} = \{\varphi_t \mid t \in \mathbb{R}_{\geq 0}\}$$

form a basis of the $\mathbb{R}$-linear space $\mathbb{P}$.

Proof. According to Definition 43, $\mathcal{B}$ spans $\mathbb{P}$. It remains to show that $\mathcal{B}$ is linearly independent.

Assume there is a nontrivial linear dependence between elements of $\mathcal{B}$. That is, there exist $t_1 < \cdots < t_n$ and $a_1, \ldots, a_n \neq 0$ such that:

$$0 = \sum_{i=1}^{n} a_i \varphi_{t_i}$$

Let $f \in \mathbb{P}$ be the right-hand side. Evaluating f at t_1 and applying Lemma 44 leads to:

$$0 = f(t_1) = \sum_{i=1}^{1} a_i = a_1$$

which contradicts the assumtion. Hence, $\mathcal{B}$ is linearly independent and thus a basis. $\square$

Motivation. In Definition 43 we equipped $\mathbb{P}$ with a partial order $\leq$. This serves two important practical needs:

1. To verify that a schedule doesn't release more resources than it uses, we check that its profile $f \in \mathbb{P}$ satisfies $f \geq 0$.
2. To verify that a schedule's resource usage doesn't exceed $r \in \mathbb{R}_{\geq 0}$, we check that its profile $f \in \mathbb{P}$ satisfies $f \leq r \cdot \varphi_0$.

Since it is so useful, we will have a closer look at how to actually check the $\leq$ relation within our algebra. Note that it is sufficient to know how to check $f \geq 0$, because every other relation $f \geq g$ can be checked by calculating $f - g$ and checking if $(f - g) \geq 0$.

Lemma 47. *For every $f \in \mathbb{P}$, $f = \sum_{i=1}^{n} a_i \varphi_{t_i}$ with $t_1 < \cdots < t_n$, nonnegativity can be verified as follows:*

$$ f \geq 0 \quad \Longleftrightarrow \quad \sum_{i=1}^{k} a_i \geq 0 \quad \forall k = 1, \ldots, n $$

Proof. This is a direct consequence of Corollary 45:

$$
\begin{aligned}
\{f(t) \mid t \in \mathbb{R}\} \quad &= \quad \left\{ \sum_{i=1}^{k} a_i \;\middle|\; k = 0, \ldots, n \right\} \\
&= \quad \{0\} \cup \left\{ \sum_{i=1}^{k} a_i \;\middle|\; k = 1, \ldots, n \right\}
\end{aligned}
$$

That is, the sums $\sum_{i=1}^{k} a_i$ describe exactly all function values $f(t)$, so it is necessary and sufficient to check that those are nonnegative. We omit the check of the zero value (for $k = 0$), as we already know it is nonnegative. $\square$

Lemma 48. *The $\leq$ relation satisfies the following properties for all $t, u \in \mathbb{R}_{\geq 0}$:*

$$ \varphi_t \geq 0 \tag{10} $$

$$ \varphi_t \geq \varphi_u \quad \text{for } t \leq u \tag{11} $$

Proof. These properties are quite obvious from Definition 42 by looking at the underlying indicator functions for φ_t and φ_u. However, this lemma can also be proved purely within the algebraic framework, by applying Lemma 47.

Consider the following profiles $f, g \in \mathbb{P}$:

$$
\begin{aligned}
f &= 1 \cdot \varphi_t \\
g &= 1 \cdot \varphi_t + (-1) \cdot \varphi_u
\end{aligned}
$$

Applying Lemma 47 to f, we know that $f \geq 0$ because $1 \geq 0$. This proves (10).

We then observe that for $t = u$, (11) is obviously true. So we assume $t < u$. This enables us to apply Lemma 47 to g. We conclude that $g \geq 0$ because $1 \geq 0$ and $1 + (-1) \geq 0$. So $\varphi_t - \varphi_u \geq 0$, which proves (11). $\qquad\square$

3.2. Shift, Duration and Final Value

Definition 49 (Shift operation)**.** For every $f \in \mathbb{R}^{\mathbb{R}}$ and $a \in \mathbb{R}_{\geq 0}$ we define $f \gg a$ ("f shifted by a") to be the following function:

$$f \gg a \colon \mathbb{R} \to \mathbb{R}$$
$$f \gg a \colon t \mapsto f(t - a)$$

The shift operator has a stronger binding than addition, such that $f \gg a + g \gg b$ means $(f \gg a) + (g \gg b)$.

Lemma 50. *For all $f, g \in \mathbb{R}^{\mathbb{R}}$, $a, b \in \mathbb{R}_{\geq 0}$ and $v \in \mathbb{R}$, the following identities hold:*

$$0 \gg a = 0 \tag{12}$$
$$f \gg 0 = f \tag{13}$$
$$(f \gg a) \gg b = f \gg (a + b) \tag{14}$$
$$\varphi_a \gg b = \varphi_{a+b} \tag{15}$$
$$(f + g) \gg a = f \gg a + g \gg a \tag{16}$$
$$(v \cdot f) \gg a = v \cdot (f \gg a) \tag{17}$$
$$f \gg a \leq g \gg a \qquad \text{for } f \leq g \tag{18}$$

Proof. For all $t \in \mathbb{R}$, we have:

$$[0 \gg a](t) = [0](t - a) = 0$$
$$[f \gg 0](t) = f(t - 0) = f(t)$$
$$[(f \gg a) \gg b](t) = [f \gg a](t - b) = f((t - b) - a) = f(t - (a + b)) = [f \gg (a + b)](t)$$
$$[\varphi_a \gg b](t) = \varphi_a(t - b) = \chi_{[a,\infty)}(t - b) = \chi_{[a+b,\infty)}(t) = \varphi_{a+b}(t)$$
$$[(f + g) \gg a](t) = [f + g](t - a) = f(t - a) + g(t - a) = [f \gg a](t) + [g \gg a](t)$$
$$[(v \cdot f) \gg a](t) = [v \cdot f](t - a) = v \cdot f(t - a) = v \cdot [f \gg a](t) = [v \cdot (f \gg a)](t)$$

Also, we have:

$$f \leq g$$
$$\implies \qquad f(t) \leq g(t) \qquad \forall t \in \mathbb{R}$$
$$\implies \qquad f(t - a) \leq g(t - a) \qquad \forall t \in \mathbb{R}$$
$$\implies \qquad [f \gg a](t) \leq [g \gg a](t) \qquad \forall t \in \mathbb{R}$$
$$\implies \qquad (f \gg a) \leq (g \gg a) \qquad\qquad\qquad\square$$

Lemma 51 (Shift closure)**.** $\mathbb{P}$ *is closed under* $\gg$. *That is, for each* $f \in \mathbb{P}$ *and* $a \in \mathbb{R}_{\geq 0}$,

$$(f \gg a) \in \mathbb{P}$$

Proof. For each $a \in \mathbb{R}_{\geq 0}$, consider the following map:

$$m_a \colon \mathbb{R}^{\mathbb{R}} \to \mathbb{R}^{\mathbb{R}}$$
$$m_a \colon f \mapsto f \gg a$$

First, m_a maps the base $\mathcal{B}$ of $\mathbb{P}$ into itself by (15). Also, m_a is $\mathbb{R}$-linear according to (16) and (17). Hence, $\mathbb{P}$ is closed under m_a. $\qquad\square$

Definition 52 (Duration and final value)**.** For every $f \in \mathbb{P}$ we define the *duration of f*, denoted as $|f|$, to be:

$$|f| = \begin{cases} 0 & f = 0 \\ t_n & f = a_1\varphi_{t_1} + \cdots + a_n\varphi_{t_n} \text{ with } t_1 < \cdots < t_n \text{ and } a_n \neq 0 \end{cases}$$

Also, we say that $f(|f|)$ is the *final value* of f.

Remark. In other words, $|f|$ is the time of the last event that affects the profile. If f is the profile of a schedule where all used resources are released at the end, the final value is $f(|f|) = 0$.

Lemma 53. *For all* $f, g \in \mathbb{P}, v \in \mathbb{R}, a \in \mathbb{R}_{\geq 0}$, *the following statements hold:*

$$|\varphi_a| = a \tag{19}$$
$$|f + g| \leq \max(|f|, |g|) \tag{20}$$
$$|v \cdot f| = |f| \qquad\qquad \text{for } v \neq 0 \tag{21}$$
$$|f \gg a| \leq |f| + a \tag{22}$$
$$|f \gg a| = |f| + a \qquad\qquad \text{for } f \neq 0 \tag{23}$$

Proof. First we get the trivial cases out of the way:

- Representing $\varphi_a = 1 \cdot \varphi_a$, we see that (19) follows directly from Definition 52.
- If $f = 0$ or $g = 0$, then (20) is obviously true.
- If $f = 0$, then (21) and (22) are obviously true.

It remains to show that (20) to (23) hold for $f \neq 0$ and $g \neq 0$, which allows us to represent f and g as follows (largest terms underlined):

$$f = a_1 \cdot \varphi_{t_1} + \cdots + \underline{a_n \cdot \varphi_{t_n}} \qquad \text{with } t_1 < \cdots < t_n \quad \text{and } a_n \neq 0$$
$$g = b_1 \cdot \varphi_{u_1} + \cdots + \underline{b_m \cdot \varphi_{u_m}} \qquad \text{with } u_1 < \cdots < u_m \text{ and } b_m \neq 0$$

So $|f| = t_n$ and $|g| = u_m$. Since $v \neq 0$ and $a_n \neq 0$, we have:

$$va_n \neq 0$$

This enables us to prove (21):

$$|v \cdot f| = \left| va_1 \cdot \varphi_{t_1} + \cdots + \underline{va_n \cdot \varphi_{t_n}} \right| = t_n = |f|$$

Since $t_1 < \cdots < t_n$, we have:

$$t_1 + a < \cdots < t_n + a$$

This enables us to prove (22) and (23):

$$|f \gg a| = \left| a_1 \cdot \varphi_{t_1+a} + \cdots + \underline{a_n \cdot \varphi_{t_n+a}} \right| = t_n + a = |f| + a$$

For (20), we distinguish the following cases:[5]

$$
\begin{array}{llll}
t_n > u_m & \implies & |f + g| = \left| \cdots + \underline{a_n \cdot \varphi_{t_n}} \right| & = t_n \\
t_n < u_m & \implies & |f + g| = \left| \cdots + \underline{b_m \cdot \varphi_{u_m}} \right| & = u_m \\
t_n = u_m \text{ and } a_n + b_m \neq 0 & \implies & |f + g| = \left| \cdots + \underline{(a_n + b_m) \cdot \varphi_{t_n}} \right| & = t_n \\
t_n = u_m \text{ and } a_n + b_m = 0 & \implies & |f + g| = \left| \cdots + \underline{0 \cdot \varphi_{t_n}} \right| & \leq t_n
\end{array}
$$

In all cases it follows that:

$$|f + g| \leq \max(t_n, u_m) = \max(|f|, |g|) \qquad \qquad \square$$

3.3. Complete Profiles

Motivation. When working with our algebra, (20) of Lemma 53 is somewhat unsatisfactory:

$$|f + g| \leq \max(|f|, |g|)$$

First, this is merely an inequality and doesn't provide a way to calculate the exact duration of two added profiles. Second, when two schedules run in parallel, we expect that their total duration doesn't shrink. So it runs against our intuition that there exist profiles f and g such that

$$|f + g| < \max(|f|, |g|)$$

---

[5] Note that in the last case, $\leq$ cannot be replaced with $<$. When the last terms cancel out, we may run into the special case of the zero profile. This effect leads to equality whenever $t_n = 0$. For example, consider $f = 2\varphi_0$ and $g = -2\varphi_0$. Here, $t_n = 0$, $u_m = 0$, $a_n = 2$, $b_m = -2$ and $f + g = 0$. So we are in the case $t_n = u_m$ and $a_n + a_m = 0$, yet we don't have $|f + g| < t_n$, but $|f + g| = t_n$.

For these reasons we introduce $\mathbb{P}^*$, the set of *complete profiles*, which is a convenient subset of $\mathbb{P}$ where the desired equality always holds:

$$|f + g| = \max(|f|, |g|)$$

However, this comes at a price, as $\mathbb{P}^*$ is not closed under all operations. So when calculating with complete profiles, we must be aware that some intermediate results may slip into $\mathbb{P} \setminus \mathbb{P}^*$. To get the best of both worlds, we try to stay within $\mathbb{P}^*$ for as long as possible, but leave it when necessary.

Definition 54 (Complete profiles). We say that a profile $f \in \mathbb{P}$ is *complete* if it is nonnegative and its final value is zero:

$$f \geq 0 \quad \text{and} \quad f(|f|) = 0$$

We define $\mathbb{P}^*$ to be the set of complete profiles, that is:

$$\mathbb{P}^* = \{f \in \mathbb{P} \mid f \geq 0 \text{ and } f(|f|) = 0\}$$

Remark. These two conditions are very natural and satisfied by any profile that corresponds to a real schedule.

1. $f \geq 0$ means that there are never more resources released than used.
2. $f(|f|) = 0$ means that all used resources are released at the end.

Profiles that fail to meet these conditions are considered *incomplete* in the sense that they seem to describe only a part of a schedule.

1. If $f \not\geq 0$, there is a time range of negative resource usage. That is, there are more resources released than actually in use. We say that f is incomplete, because it seems the *first part* of the schedule is missing, containing events that start using those resources.
2. If $f \geq 0$, but $f(|f|) \neq 0$, the profile ends with a positive resource usage, using those forever. We say that f is incomplete, because it seems the *second part* of the schedule is missing, containing events that finish using those resources.

Lemma 55. *For all $f, g \in \mathbb{P}^*$ the duration of their sum can be calculated via:*

$$|f + g| = \max(|f|, |g|) \tag{24}$$

Proof. If $f = 0$ or $g = 0$, then (24) is obviously true. If $f \neq 0$ and $g \neq 0$, we can represent f and g as follows (largest terms underlined):

$$f = a_1 \cdot \varphi_{t_1} + \cdots + \underline{a_n \cdot \varphi_{t_n}} \quad \text{with } t_1 < \cdots < t_n \quad \text{and } a_n \neq 0$$
$$g = b_1 \cdot \varphi_{u_1} + \cdots + \underline{b_m \cdot \varphi_{u_m}} \quad \text{with } u_1 < \cdots < u_m \text{ and } b_m \neq 0$$

So $|f| = t_n$ and $|g| = u_m$. It follows that $f(t_n) = f(|f|) = 0$.

We know that $n \neq 1$, because for $n = 1$ we would have $0 = f(t_n) = f(t_1) = a_1 = a_n$, which would contradict $a_n \neq 0$.

So $n \geq 2$, hence we can apply Lemma 44 to t_n as well as t_{n-1}:

$$f(t_n) = \sum_{i=1}^{n} a_i \qquad\qquad f(t_{n-1}) = \sum_{i=1}^{n-1} a_i$$

Using $f \geq 0$, it follows that:

$$a_n = \sum_{i=1}^{n} a_i - \sum_{i=1}^{n-1} a_i = f(t_n) - f(t_{n-1}) = -f(t_{n-1}) \leq 0$$

Since $a_n \neq 0$, we conclude $a_n < 0$. With similar reasoning, we conclude $b_m < 0$. Hence,

$$a_n + b_m \neq 0$$

Finally, we distinguish the following cases:

$$
\begin{aligned}
t_n > u_m &\implies & |f + g| = \left| \cdots + \underline{a_n \cdot \varphi_{t_n}} \right| & & = t_n & = \max(t_n, u_m) \\
t_n < u_m &\implies & |f + g| = \left| \cdots + \underline{b_m \cdot \varphi_{u_m}} \right| & & = u_m & = \max(t_n, u_m) \\
t_n = u_m &\implies & |f + g| = \left| \cdots + \underline{(a_n + b_m) \cdot \varphi_{t_n}} \right| & = t_n & & = \max(t_n, u_m)
\end{aligned}
$$

In all cases it follows that:

$$|f + g| = \max(t_n, u_m) = \max(|f|, |g|) \hspace{3cm} \square$$

Lemma 56. $\mathbb{P}^*$ *is closed under conical combination[6] and shift operation. That is,*

$$f + g \in \mathbb{P}^* \tag{25}$$
$$v \cdot f \in \mathbb{P}^* \quad \text{for } v \geq 0 \tag{26}$$
$$f \gg a \in \mathbb{P}^* \tag{27}$$

for all $f, g \in \mathbb{P}^*$, $v \in \mathbb{R}$ and $a \in \mathbb{R}_{\geq 0}$.

---

[6] A *conical combination* is a linear combination where all coefficients are nonnegative. [Jet86, p. 51]

Proof. According to Definition 54, we must prove the following statements:

$$f + g \geq 0$$
$$v \cdot f \geq 0$$
$$f \gg a \geq 0$$
$$[f + g](|f + g|) = 0$$
$$[v \cdot f](|v \cdot f|) = 0$$
$$[f \gg a](|f \gg a|) = 0$$

Since $f \geq 0$ and $g \geq 0$, it follows that:

$$f + g \geq 0$$

Since $v \geq 0$ and $f \geq 0$, it follows that:

$$v \cdot f \geq 0$$

According to (12) and (18), $f \geq 0$ implies:

$$(f \gg a) \geq (0 \gg a) = 0$$

Without loss of generality, we assume $|f| \geq |g|$. It follows that:

$$g(|f|) = g(|g|) = 0$$
$$|f + g| = \max(|f|, |g|) = |f|$$

Hence,

$$[f + g](|f + g|) = [f + g](|f|) = f(|f|) + g(|f|) = 0 + 0 = 0$$

For $v = 0$ we have:

$$[v \cdot f](|v \cdot f|) = [0 \cdot f](|v \cdot f|) = [0](|v \cdot f|) = 0$$

For $v > 0$ we apply (21):

$$[v \cdot f](|v \cdot f|) = [v \cdot f](|f|) = v \cdot f(|f|) = v \cdot 0 = 0$$

For $f = 0$ we apply (12):

$$[f \gg a](|f \gg a|) = [0 \gg a](|f \gg a|) = [0](|f \gg a|) = 0$$

For $f \neq 0$ we apply (23):

$$[f \gg a](|f \gg a|) = [f \gg a](|f| + a) = f(|f| + a - a) = f(|f|) = 0 \qquad \square$$

Lemma 57. *All finite interval functions in the nonnegative range that are closed to the left and open to the right are complete profiles. That is, for all $a, b \in \mathbb{R}_{\geq 0}$ with $a \leq b$:*

$$\chi_{[a,b)} = \varphi_a - \varphi_b \in \mathbb{P}^* \tag{28}$$

Proof. For $a = b$, we have:

$$\chi_{[a,b)} = \chi_\varnothing = 0 \in \mathbb{P}^*$$

For $a < b$, we first note that $\chi_{[a,b)}$ is a profile, because it is a linear combination of primitive functions:

$$\chi_{[a,b)} = 1 \cdot \varphi_a + (-1) \cdot \varphi_b \in \mathbb{P}$$

Since the image of $\chi_{[a,b)}$ is $\{0, 1\}$, it follows that:

$$\chi_{[a,b)} \geq 0$$

From $a < b$ and Definition 52 it follows that $\left|\chi_{[a,b)}\right| = |\varphi_a - \varphi_b| = b$, hence:

$$\chi_{[a,b)}\left(\left|\chi_{[a,b)}\right|\right) = \chi_{[a,b)}(b) = 0$$

So by Definition 54, $\chi_{[a,b)} \in \mathbb{P}^*$. □

Lemma 58. *The complete profiles are the conical hull[7] of all finite interval functions in the nonnegative range that are closed to the left and open to the right. That is,*

$$\mathbb{P}^* = \mathrm{coni}(\mathcal{I})$$

where

$$\mathcal{I} = \left\{\chi_{[a,b)} \mid a, b \in \mathbb{R}_{\geq 0} \text{ with } a < b\right\}$$

Proof. From Lemma 57 it follows that:

$$\mathcal{I} \subseteq \mathbb{P}^*$$

According to Lemma 56, $\mathbb{P}^*$ is closed under conical combination, so we conclude:

$$\mathrm{coni}(\mathcal{I}) \subseteq \mathbb{P}^*$$

It remains to show that $\mathbb{P}^* \subseteq \mathrm{coni}(\mathcal{I})$. Let $f \in \mathbb{P}^*$ and represent it as follows:

$$f = \sum_{i=1}^{n} a_i \varphi_{t_i} \quad \text{with } t_1 < \cdots < t_n$$

---

[7]The *conical hull* $\mathrm{coni}(S)$ is the set of all conical combinations of elements of S. [Jet86, p. 51]

Further, let $0, b_1, \ldots, b_n$ be the function values according to Lemma 44. That is,

$$b_k = \sum_{i=1}^{k} a_i \qquad \text{for } k = 1, \ldots, n$$

Now, f can be written as:

$$f(t) = \begin{cases} 0 & t \in (-\infty, t_1) \\ b_1 & t \in [t_1, t_2) \\ \cdots \\ b_{n-1} & t \in [t_{n-1}, t_n) \\ b_n & t \in [t_n, \infty) \end{cases}$$

This translates directly to a linear combination of interval functions:

$$f = 0 \cdot \chi_{(-\infty, t_1)} + b_1 \cdot \chi_{[t_1, t_2)} + \cdots + b_{n-1} \cdot \chi_{[t_{n-1}, t_n)} + b_n \cdot \chi_{[t_n, \infty)}$$

Since $b_n = f(t_n) = f(|t|) = 0$, this simplifies to:

$$f = b_1 \cdot \chi_{[t_1, t_2)} + \cdots + b_{n-1} \cdot \chi_{[t_{n-1}, t_n)}$$

Since $f \geq 0$, we know that all $b_k \geq 0$. So this is not just a linear combination, but a conical combination. Hence,

$$f \in \mathrm{coni}(\mathcal{I})$$

This proves $\mathbb{P}^* \subseteq \mathrm{coni}(\mathcal{I})$, so in total we have $\mathbb{P}^* = \mathrm{coni}(\mathcal{I})$. $\qquad \square$

3.4. Arrangement

Motivation. After having defined all basic operations on profiles, we want to calculate the profile for combined schedules as of Definition 38. However, that operation lacks useful properties such as associativity. So we introduce an intermediate operation that arranges two profiles exactly as needed, but does not prepend the additional operations (checkpoint holding and forward calculation), as illustrated in Figure 8. This *arrangement* operation has some nice properties and will serve as building block for the real combination of schedule profiles.

Definition 59 (Arrangement). For two profiles $f, g \in \mathbb{P}$ we define f *arranged with* g, denoted as $f * g$, to be:

$$f * g = f \gg \frac{|g|}{2} + g \gg |f|$$
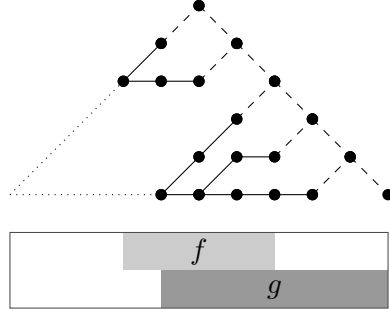
Figure 8.: Arrangement.

Lemma 60. $\mathbb{P}^*$ *is closed under arrangement. That is, for all $f, g \in \mathbb{P}^*$ we have:*

$$f * g \in \mathbb{P}^*$$

Proof. Arrangement is just a composition of two shifts and one addition, and $\mathbb{P}^*$ is closed under shift and addition (Lemma 56). So $\mathbb{P}^*$ is also closed under arrangement. $\square$

Lemma 61. *Arrangement has the following properties for all $f, g \in \mathbb{P}, v \in \mathbb{R}, a \in \mathbb{R}_{\geq 0}$:*

$$0 * f = f \tag{29}$$
$$f * 0 = f \tag{30}$$
$$v \cdot (f * g) = (v \cdot f) * (v \cdot g) \tag{31}$$
$$(f * g) \gg a = (f \gg a) * g \qquad \text{for } f \neq 0 \tag{32}$$

Moreover, for all $f, g, h \in \mathbb{P}^*$:

$$(f + g) * h = f * h + g \gg \frac{|h|}{2} \qquad \text{for } |f| \geq |g| \tag{33}$$
$$h * (f + g) = h * f + g \gg |h| \qquad \text{for } |f| \geq |g| \tag{34}$$
$$|f * g| = |f| + |g| \tag{35}$$
$$(f * g) * h = f * (g * h) \tag{36}$$

Proof. Let $f, g \in \mathbb{P}, v \in \mathbb{R}, a \in \mathbb{R}_{\geq 0}$. We apply (12) and (13) to prove (29) and (30):

$$0 * f = 0 \gg \frac{|f|}{2} + f \gg |0| = 0 + (f \gg 0) = f$$
$$f * 0 = f \gg \frac{|0|}{2} + 0 \gg |f| = (f \gg 0) + 0 = f$$

We apply (29) to prove (31) in the case $v = 0$:

$$0 \cdot (f * g) = 0 = 0 * 0 = (0 \cdot f) * (0 \cdot g)$$

We apply (17) and (21) to prove (31) in the case $v \neq 0$:

$$v \cdot (f * g) = v \cdot \left(f \gg \frac{|g|}{2} + g \gg |f| \right)$$

$$= v \cdot \left(f \gg \frac{|g|}{2} \right) + v \cdot (g \gg |f|)$$

$$= (v \cdot f) \gg \frac{|g|}{2} + (v \cdot g) \gg |f|$$

$$= (v \cdot f) \gg \frac{|v \cdot g|}{2} + (v \cdot g) \gg |v \cdot f|$$

$$= (v \cdot f) * (v \cdot g)$$

Assuming $f \neq 0$, we apply (16), (14) and (23) to prove (32):

$$(f * g) \gg a = \left(f \gg \frac{|g|}{2} + g \gg |f| \right) \gg a$$

$$= \left(f \gg \frac{|g|}{2} \right) \gg a + (g \gg |f|) \gg a$$

$$= f \gg \left(\frac{|g|}{2} + a \right) + g \gg (|f| + a)$$

$$= (f \gg a) \gg \frac{|g|}{2} + g \gg |f \gg a|$$

$$= (f \gg a) * g$$

From now on, let $f, g, h \in \mathbb{P}^*$.

For $|f| \geq |g|$, we apply (16) and (24) to prove (33) and (34):

$$(f + g) * h = (f + g) \gg \frac{|h|}{2} + h \gg |f + g|$$

$$= f \gg \frac{|h|}{2} + g \gg \frac{|h|}{2} + h \gg \max(|f|, |g|)$$

$$= f \gg \frac{|h|}{2} + g \gg \frac{|h|}{2} + h \gg |f|$$

$$= f * h + g \gg \frac{|h|}{2}$$

$$h * (f + g) = h \gg \frac{|f + g|}{2} + (f + g) \gg |h|$$

$$= h \gg \frac{\max(|f|, |g|)}{2} + f \gg |h| + g \gg |h|$$

$$= h \gg \frac{|f|}{2} + f \gg |h| + g \gg |h|$$

$$= h * f + g \gg |h|$$

44

We apply (30) and (29) to prove (35) and (36) for the case $g = 0$:

$$|f * 0| = |f| = |f| + |0|$$
$$(f * 0) * h = f * h = f * (0 * h)$$

For $g \neq 0$, we apply (22) and (23) to prove a helper inequality:

$$\left| f \gg \frac{|g|}{2} \right| \leq |f| + \frac{|g|}{2} \leq |g| + |f| = |g \gg |f||$$

Using that, we apply (24) and (23) to prove (35) for $g \neq 0$:

$$|f * g| = \left| f \gg \frac{|g|}{2} + g \gg |f| \right|$$
$$= \max \left(\left| f \gg \frac{|g|}{2} \right|, |g \gg |f|| \right)$$
$$= |g \gg |f||$$
$$= |g| + |f|$$

Finally, we apply (33), (32), (14) and (35) to prove (36) for $g \neq 0$:

$$(f * g) * h = \left(f \gg \frac{|g|}{2} + g \gg |f| \right) * h$$
$$= (g \gg |f|) * h + \left(f \gg \frac{|g|}{2} \right) \gg \frac{|h|}{2}$$
$$= (g * h) \gg |f| + f \gg \frac{|g| + |h|}{2}$$
$$= (g * h) \gg |f| + f \gg \frac{|g * h|}{2}$$
$$= f * (g * h) \qquad \qquad \square$$

Remark. The associativity allows us to omit the parentheses as long as we stay in $\mathbb{P}^*$:

$$f * g * \cdots * h \in \mathbb{P}^*$$

Remark. We cannot generalize (33) to (36) from $\mathbb{P}^*$ to $\mathbb{P}$. For example, consider:

$$f = \varphi_0 + \varphi_1$$
$$g = -\varphi_0$$
$$h = \varphi_0$$

This would be a counterexample to (35), because:

$$|f * g| = 0$$
$$|f| + |g| = 1$$

It would also be a counterexample to (36), because:

$$(f * g) * h = 2\varphi_0$$
$$f * (g * h) = \varphi_0 + \varphi_1$$

3.5. Algebra Summary

Remark. Up to now, we built a fairly comprehensive algebra. We introduced new notations and rooted them in the theory of linear algebra and partially ordered sets. We proved that almost everything fits together neatly, and drew clear boundaries around the issues that didn't work out as nicely as we may have hoped. After all that, it is time to take one step back and look at what we've got so far.

Throughout this summary we will use the following variables:

$$a, b \in \mathbb{R}_{\geq 0}$$
$$v, w \in \mathbb{R}$$
$$f, g, h \in \mathbb{P}$$

We introduce the following primitives, operations and relations:

$0 \in \mathbb{P}$	Definition 43
$\varphi_a \in \mathbb{P}$	Definition 42
$f + g \in \mathbb{P}$	Definition 43
$v \cdot f \in \mathbb{P}$	Definition 43
$f \leq g \in \mathbb{B}$	Definition 43
$f \gg a \in \mathbb{P}$	Definition 49 and Lemma 51
$|f| \in \mathbb{R}_{\geq 0}$	Definition 52
$\mathbb{P}^* \in 2^{\mathbb{P}}$	Definition 54
$f * g \in \mathbb{P}$	Definition 59

These satisfy the following rules within $\mathbb{P}$:

$0 + f = f$	Definition 43
$f + g = g + f$	Definition 43
$(f + g) + h = f + (g + h)$	Definition 43
$0 \cdot f = 0$	Definition 43

$$1 \cdot f = f \qquad\qquad \text{Definition 43}$$
$$v \cdot (w \cdot f) = (vw) \cdot f \qquad\qquad \text{Definition 43}$$
$$(v + w) \cdot f = (v \cdot f) + (w \cdot f) \qquad\qquad \text{Definition 43}$$
$$\varphi_a \geq 0 \qquad\qquad \text{Lemma 48}$$
$$\varphi_a \geq \varphi_b \qquad \text{for } a \leq b \qquad \text{Lemma 48}$$
$$f + h \leq g + h \qquad \text{for } f \leq g \qquad \text{Definition 43}$$
$$a \cdot f \geq 0 \qquad \text{for } f \geq 0 \qquad \text{Definition 43}$$
$$0 \gg a = 0 \qquad\qquad \text{Lemma 50}$$
$$f \gg 0 = f \qquad\qquad \text{Lemma 50}$$
$$(f \gg a) \gg b = f \gg (a + b) \qquad\qquad \text{Lemma 50}$$
$$\varphi_a \gg b = \varphi_{a+b} \qquad\qquad \text{Lemma 50}$$
$$(f + g) \gg a = f \gg a + g \gg a \qquad\qquad \text{Lemma 50}$$
$$(v \cdot f) \gg a = v \cdot (f \gg a) \qquad\qquad \text{Lemma 50}$$
$$f \gg a \leq g \gg a \qquad \text{for } f \leq g \qquad \text{Lemma 50}$$
$$|\varphi_a| = a \qquad\qquad \text{Lemma 53}$$
$$|f + g| \leq \max(|f|, |g|) \qquad\qquad \text{Lemma 53}$$
$$|v \cdot f| = |f| \qquad \text{for } v \neq 0 \qquad \text{Lemma 53}$$
$$|f \gg a| \leq |f| + a \qquad\qquad \text{Lemma 53}$$
$$|f \gg a| = |f| + a \qquad \text{for } f \neq 0 \qquad \text{Lemma 53}$$
$$f * g = f \gg \frac{|g|}{2} + g \gg |f| \qquad\qquad \text{Definition 59}$$
$$0 * f = f \qquad\qquad \text{Lemma 61}$$
$$f * 0 = f \qquad\qquad \text{Lemma 61}$$
$$v \cdot (f * g) = (v \cdot f) * (v \cdot g) \qquad\qquad \text{Lemma 61}$$
$$(f * g) \gg a = (f \gg a) * g \qquad \text{for } f \neq 0 \qquad \text{Lemma 61}$$

And they satisfy the following additional rules for $f, g, h \in \mathbb{P}^*$:

$$0 \in \mathbb{P}^* \qquad \qquad \qquad \text{Definition 54}$$
$$\varphi_a - \varphi_b \in \mathbb{P}^* \qquad \text{for } a \leq b \qquad \text{Lemma 57}$$
$$f + g \in \mathbb{P}^* \qquad \qquad \qquad \text{Lemma 56}$$
$$a \cdot f \in \mathbb{P}^* \qquad \qquad \qquad \text{Lemma 56}$$
$$f \gg a \in \mathbb{P}^* \qquad \qquad \qquad \text{Lemma 56}$$
$$f * g \in \mathbb{P}^* \qquad \qquad \qquad \text{Lemma 60}$$
$$f \geq 0 \qquad \qquad \qquad \text{Definition 54}$$
$$f(|f|) = 0 \qquad \qquad \qquad \text{Definition 54}$$
$$|f + g| = \max(|f|, |g|) \qquad \qquad \text{Lemma 55}$$
$$(f + g) * h = f * h + g \gg \frac{|h|}{2} \quad \text{for } |f| \geq |g| \qquad \text{Lemma 61}$$
$$h * (f + g) = h * f + g \gg |h| \quad \text{for } |f| \geq |g| \qquad \text{Lemma 61}$$
$$|f * g| = |f| + |g| \qquad \qquad \qquad \text{Lemma 61}$$
$$(f * g) * h = f * (g * h) \qquad \qquad \text{Lemma 61}$$

3.6. Schedule Profiles

Motivation. Now that we built our algebra, it is time to apply it to reversal schedules. We start with the elementary schedule ε and continue with the composition of schedules. We will simultaneously consider processes profiles as well as resources profiles.

Definition 62 (Elementary profile)**.** We define the elementary profile e to be:

$$e \in \mathbb{P}^*$$
$$e = \varphi_0 - \varphi_2$$

Lemma 63. *The profile e describes the processes profile as well as the resources profile of the elementary schedule ε. That is:*

$$\operatorname{procp} \varepsilon = e$$
$$\operatorname{resp} \varepsilon = e$$

Proof. Schedule ε starts with a preparation step from time 0 to time 1, that is, $\varphi_0 - \varphi_1$. This is followed by a reversal step from time 1 to time 2, that is, $\varphi_1 - \varphi_2$. Their sum is $(\varphi_0 - \varphi_1) + (\varphi_1 - \varphi_2) = \varphi_0 - \varphi_2 = e$. $\qquad\square$

Lemma 64 (Profile composition)**.** *The* composition *of two profiles $f, g \in \mathbb{P}$ is one of the following operations, depending on whether we deal with processes or resources profiles:*

$$[f, g] = f * g + \varphi_0 - \varphi_{\frac{|g|}{2}} \qquad \text{(processes)}$$

$$(f, g) = f * g + 2\varphi_0 - \varphi_{\frac{|g|}{2}} - \varphi_{|f|} \qquad \text{(resources)}$$

That is, for all schedules S and T we have:

$$\text{procp}\,(S, T) = [\text{procp}\,S, \text{procp}\,T]$$

$$\text{resp}\,(S, T) = (\text{resp}\,S, \text{resp}\,T)$$

Proof. To understand how this describes the composition of schedules, we must expand the arrangement operation $f * g$ and reorder the summands:

$$[f, g] = \left(\varphi_0 - \varphi_{\frac{|g|}{2}} + f \gg \frac{|g|}{2} \right) + g \gg |f|$$

$$(f, g) = \left(\varphi_0 - \varphi_{\frac{|g|}{2}} + f \gg \frac{|g|}{2} \right) + \left(\varphi_0 - \varphi_{|f|} + g \gg |f| \right)$$

These expressions correspond directly to the following natural language description of Definition 38:

- For the processes profile $[f, g]$, we count the forward steps from time 0 to time $\frac{|g|}{2}$, followed by f starting at time $\frac{|g|}{2}$. Concurrently, g starts at time $|f|$.
- The resources profile (f, g) is almost identical, except that we also count the checkpoint holding from time 0 to time $|f|$ before starting g at time $|f|$.

$\square$

Motivation. Since we are dealing with profiles of real schedules, we expect the composition operations to preserve complete profiles. Also, we expect the total duration to be the sum of the individual durations. Just to be sure, we show both properties formally.

Lemma 65. $\mathbb{P}^*$ *is closed under composition. That is, for all $f, g \in \mathbb{P}^*$:*

$$[f, g] \in \mathbb{P}^*$$

$$(f, g) \in \mathbb{P}^*$$

Proof. First, we apply Lemma 57 to $0 \le \frac{|g|}{2}$ and $0 \le |f|$:

$$\varphi_0 - \varphi_{\frac{|g|}{2}} \in \mathbb{P}^*$$

$$\varphi_0 - \varphi_{|f|} \in \mathbb{P}^*$$

According to Lemmas 56 and 60, we know that $\mathbb{P}^*$ is closed under addition and arrangement. Hence,

$$[f,g] = f * g + \left(\varphi_0 - \varphi_{\frac{|g|}{2}}\right) \qquad\qquad \in \mathbb{P}^*$$
$$(f,g) = f * g + \left(\varphi_0 - \varphi_{\frac{|g|}{2}}\right) + \left(\varphi_0 - \varphi_{|f|}\right) \in \mathbb{P}^* \qquad\qquad \square$$

Lemma 66. *For all $f, g \in \mathbb{P}^*$:*

$$|[f,g]| = |(f,g)| = |f| + |g| \tag{37}$$

Proof. This is a direct application of (24) and (35):

$$|[f,g]| = \max\left(|f * g|, 0, \frac{|g|}{2}\right) \quad = \max\left(|f| + |g|, 0, \frac{|g|}{2}\right) \quad = |f| + |g|$$
$$|(f,g)| = \max\left(|f * g|, 0, \frac{|g|}{2}, |f|\right) = \max\left(|f| + |g|, 0, \frac{|g|}{2}, |f|\right) = |f| + |g| \qquad \square$$

Lemma 67 (Profiles of small schedules). *The profile of ε is:*

$$e = \varphi_0 - \varphi_2$$
$$|e| = 2$$

The profile of $(\varepsilon, \varepsilon)$ is:

$$[e,e] = \varphi_0 + \varphi_2 - \varphi_3 - \varphi_4$$
$$(e,e) = 2\varphi_0 - \varphi_3 - \varphi_4$$
$$|\cdots| = 4$$

The profile of $((\varepsilon, \varepsilon), \varepsilon)$ is:

$$[[e,e],e] = \varphi_0 + \varphi_3 - \varphi_5 - \varphi_6$$
$$((e,e),e) = 2\varphi_0 + \varphi_1 - \varphi_4 - \varphi_5 - \varphi_6$$
$$|\cdots| = 6$$

The profile of $(\varepsilon, (\varepsilon, \varepsilon))$ is:

$$[e,[e,e]] = \varphi_0 + \varphi_2 - \varphi_5 - \varphi_6$$
$$(e,(e,e)) = 2\varphi_0 + \varphi_2 - \varphi_4 - \varphi_5 - \varphi_6$$
$$|\cdots| = 6$$

Remark. These schedules are depicted in Figure 6. Moreover, Figure 7 shows the resources profile of $(\varepsilon, (\varepsilon, \varepsilon))$.

Proof. This is just an exercise in expanding the terms by applying the definitions:

$$|e| = |\varphi_0 - \varphi_2|$$
$$= 2$$
$$[e, e] = e * e + \varphi_0 - \varphi_1$$
$$= (e \gg 1) + (e \gg 2) + \varphi_0 - \varphi_1$$
$$= (\varphi_1 - \varphi_3) + (\varphi_2 - \varphi_4) + \varphi_0 - \varphi_1$$
$$= \varphi_0 + \varphi_2 - \varphi_3 - \varphi_4$$
$$(e, e) = e * e + 2\varphi_0 - \varphi_1 - \varphi_2$$
$$= (e \gg 1) + (e \gg 2) + 2\varphi_0 - \varphi_1 - \varphi_2$$
$$= (\varphi_1 - \varphi_3) + (\varphi_2 - \varphi_4) + 2\varphi_0 - \varphi_1 - \varphi_2$$
$$= 2\varphi_0 - \varphi_3 - \varphi_4$$
$$[[e, e], e] = [e, e] * e + \varphi_0 - \varphi_1$$
$$= ([e, e] \gg 1) + (e \gg 4) + \varphi_0 - \varphi_1$$
$$= (\varphi_1 + \varphi_3 - \varphi_4 - \varphi_5) + (\varphi_4 - \varphi_6) + \varphi_0 - \varphi_1$$
$$= \varphi_0 + \varphi_3 - \varphi_5 - \varphi_6$$
$$((e, e), e) = (e, e) * e + 2\varphi_0 - \varphi_1 - \varphi_4$$
$$= ((e, e) \gg 1) + (e \gg 4) + 2\varphi_0 - \varphi_1 - \varphi_4$$
$$= (2\varphi_1 - \varphi_4 - \varphi_5) + (\varphi_4 - \varphi_6) + 2\varphi_0 - \varphi_1 - \varphi_4$$
$$= 2\varphi_0 + \varphi_1 - \varphi_4 - \varphi_5 - \varphi_6$$
$$[e, [e, e]] = e * [e, e] + \varphi_0 - \varphi_2$$
$$= (e \gg 2) + ([e, e] \gg 2) + \varphi_0 - \varphi_2$$
$$= (\varphi_2 - \varphi_4) + (\varphi_2 + \varphi_4 - \varphi_5 - \varphi_6) + \varphi_0 - \varphi_2$$
$$= \varphi_0 + \varphi_2 - \varphi_5 - \varphi_6$$
$$(e, (e, e)) = e * (e, e) + 2\varphi_0 - \varphi_2 - \varphi_2$$
$$= (e \gg 2) + ((e, e) \gg 2) + 2\varphi_0 - \varphi_2 - \varphi_2$$
$$= (\varphi_2 - \varphi_4) + (2\varphi_2 - \varphi_5 - \varphi_6) + 2\varphi_0 - \varphi_2 - \varphi_2$$
$$= 2\varphi_0 + \varphi_2 - \varphi_4 - \varphi_5 - \varphi_6 \qquad \square$$

Motivation. We clearly see that our new multiplications as of Lemma 64 are not associative, as associativity fails to hold for simples examples:

$$[e, [e, e]] \neq [[e, e], e]$$
$$(e, (e, e)) \neq ((e, e), e)$$

This is no surprise, as these decribe the profiles of $(\varepsilon, (\varepsilon, \varepsilon))$ respectively $((\varepsilon, \varepsilon), \varepsilon)$, which are different schedules with different resource usage.

So we follow the usual approach when associativity fails: We calculate the *associator* for each type of multiplication. Note that although we borrow that term from the theory

of nonassociative algebras [Sch66, p. 13], there are important differences to keep in mind:

1. Our multiplications as of Lemma 64 are not bilinear, so their associators are not linear in each argument, either. Hence, the classic results about nonassociative algebras are not applicable here. [Sch66, p. 1]
2. We calculate $[f, [g, h]] - [[f, g], h]$ instead of $[[f, g], h] - [f, [g, h]]$ to make the right-hand sides easier to understand. That is, our associators have the reverse sign.

Lemma 68 (Associators). *For all complete profiles $f, g, h \in \mathbb{P}^*$:*

$$[f, [g, h]] - [[f, g], h] = \varphi_{|f|} - \varphi_{|f| + \frac{|h|}{2}} \tag{38}$$

$$(f, (g, h)) - ((f, g), h) = \varphi_{|f|} - \varphi_{\frac{|h|}{2}} \tag{39}$$

Proof. We expand the first term using Lemma 64, (37), (34), (36), (15) and (16). The remaining terms expand in almost the same way.

$$[f, [g, h]] = f * [g, h] + \varphi_0 - \varphi_{\frac{|[g,h]|}{2}}$$

$$= f * \left(g * h + \varphi_0 - \varphi_{\frac{|h|}{2}} \right) + \varphi_0 - \varphi_{\frac{|g| + |h|}{2}}$$

$$= f * g * h + \left(\varphi_0 - \varphi_{\frac{|h|}{2}} \right) \gg |f| + \varphi_0 - \varphi_{\frac{|g| + |h|}{2}}$$

$$= f * g * h + \varphi_{|f|} - \varphi_{|f| + \frac{|h|}{2}} + \varphi_0 - \varphi_{\frac{|g| + |h|}{2}}$$

$$= f * g * h + \varphi_0 - \varphi_{\frac{|g| + |h|}{2}} + \varphi_{|f|} - \varphi_{|f| + \frac{|h|}{2}}$$

$$[[f, g], h] = f * g * h + \varphi_0 - \varphi_{\frac{|g| + |h|}{2}}$$

Subtracting both equations proves (38). Further,

$$(f, (g, h)) = f * g * h + 2\varphi_0 - \varphi_{\frac{|g| + |h|}{2}} - \varphi_{|f| + \frac{|h|}{2}} - \varphi_{|f| + |g|} + \varphi_{|f|}$$

$$((f, g), h) = f * g * h + 2\varphi_0 - \varphi_{\frac{|g| + |h|}{2}} - \varphi_{|f| + \frac{|h|}{2}} - \varphi_{|f| + |g|} + \varphi_{\frac{|h|}{2}}$$

Subtracting both equations proves (39). $\square$

Remark. In this proof, we see clearly how the introduction of the associative arrangement operator "$*$" payed off, as promised in Section 3.4.

Motivation. Our associators have a surprisingly simple form. Both are independent of g. Moreover, only the durations of f and h matter, not their exact shape. This gives rise to the main theorem in this section:

Theorem 69 (Associator inequalities)**.** *For all complete profiles $f, g, h \in \mathbb{P}^*$ with $|h| > 0$:*

$$[f, [g, h]] > [[f, g], h] \tag{40}$$

$$(f, (g, h)) \geq ((f, g), h) \quad \text{for } |f| \leq \frac{|h|}{2} \tag{41}$$

$$(f, (g, h)) < ((f, g), h) \quad \text{for } |f| > \frac{|h|}{2} \tag{42}$$

Remark. This theorem helps us to decide how to combine three schedules. Should we combine them as $(f, (g, h))$ or as $((f, g), h)$?

- If $|f| \leq \frac{|h|}{2}$, we should always use $((f, g), h)$, because that one uses fewer processes and not more resources. This is demonstrated in Figure 9.
- If $|f| > \frac{|h|}{2}$, we should consider both variants, because $((f, g), h)$ uses fewer processes, but $(f, (g, h))$ uses fewer resources.
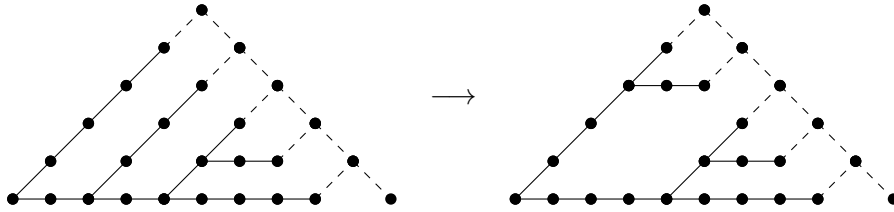
Figure 9.: Associator inequality.

Proof. Applying (11) to $|f| \leq |f| + \frac{|h|}{2}$ leads to:

$$\varphi_{|f|} \geq \varphi_{|f| + \frac{|h|}{2}}$$

Moreover, since $|h| > 0$ we know that $\varphi_{|f|} \neq \varphi_{|f| + \frac{|h|}{2}}$. Hence,

$$\varphi_{|f|} > \varphi_{|f| + \frac{|h|}{2}}$$

In the same way, we conclude:

$$\varphi_{|f|} \geq \varphi_{\frac{|h|}{2}} \quad \text{for } |f| \leq \frac{|h|}{2}$$

$$\varphi_{|f|} < \varphi_{\frac{|h|}{2}} \quad \text{for } |f| > \frac{|h|}{2}$$

Finally, we apply Lemma 68 to prove (40) to (42):

$$[f, [g, h]] - [[f, g], h] = \varphi_{|f|} - \varphi_{|f| + \frac{|h|}{2}} > 0$$

$$(f, (g, h)) - ((f, g), h) = \varphi_{|f|} - \varphi_{\frac{|h|}{2}} \quad \geq 0 \quad \text{for } |f| \leq \frac{|h|}{2}$$

$$(f, (g, h)) - ((f, g), h) = \varphi_{|f|} - \varphi_{\frac{|h|}{2}} \quad < 0 \quad \text{for } |f| > \frac{|h|}{2} \qquad \square$$

4. Optimal and Suboptimal Schedules

In this chapter we will develop a double sequence S_k^p of schedules, each of which will use no more than p processes and k resources. Ideally, every S_k^p would be optimal in the sense that it achieves the maximum reach ℓ, given p and k with $p \geq 1$ and $k \geq p$. That is, we would like to have:

$$\ell(S_k^p) = \ell_{\max}(p, k)$$

At present, however, only certain classes of optima are known. For the remaining classes suboptimal schedules are provided. Those schedules will provide a lower bound on the maximum reach:

$$\ell(S_k^p) \leq \ell_{\max}(p, k)$$

All schedules S_k^p are implemented in the tool (Appendix A).

4.1. Fibonacci Schedules

Definition 70 (Fibonacci schedules)**.** We define S_k^k to be the optimal schedules for $p = k$ developed in [Wal99]:

$$
\begin{aligned}
S_1^1 &:= \varepsilon \\
S_2^2 &:= (\varepsilon, \varepsilon) \\
S_k^k &:= \left(S_{k-2}^{k-2}, S_{k-1}^{k-1} \right) \quad \text{for } k \geq 3
\end{aligned}
$$

Example 71. Some of these schedules are shown in Figure 10.

Remark. We refer to these schedules as *Fibonacci schedules*, because they exhibit the same recursive structure as the Fibonacci numbers, and because their reaches ℓ are exactly the Fibonacci numbers (shifted by one position):

$$\ell(S_k^k) = \ell_{\max}(k, k) = F_{k+1}$$

As explained in Section 1.3, these schedules are also optimal for $p \geq \left\lceil \frac{k+1}{2} \right\rceil$, but we *will not necessarily* define $S_k^{\left\lceil \frac{k+1}{2} \right\rceil}, S_k^{\left\lceil \frac{k+1}{2} \right\rceil + 1}, \ldots, S_k^{k-1}$ to be S_k^k. Instead, we will define those S_k^p separately in the following sections, which may lead to different schedules that fit better into the systematics of the sequence for that respective p.
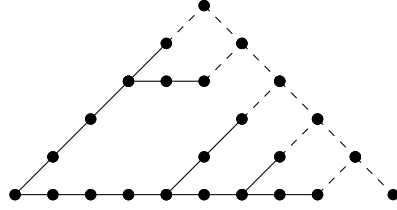
$$S_4^4 = ((\varepsilon, \varepsilon), (\varepsilon, (\varepsilon, \varepsilon)))$$

$$\text{procp } S_4^4 = \varphi_0 + \varphi_4 + \varphi_5 - \varphi_7 - \varphi_9 - \varphi_{10}$$

$$\text{resp } S_4^4 = 2\varphi_0 + \varphi_3 + \varphi_4 - \varphi_7 - \varphi_8 - \varphi_9 - \varphi_{10}$$

$$\ell(S_4^4) = 5$$
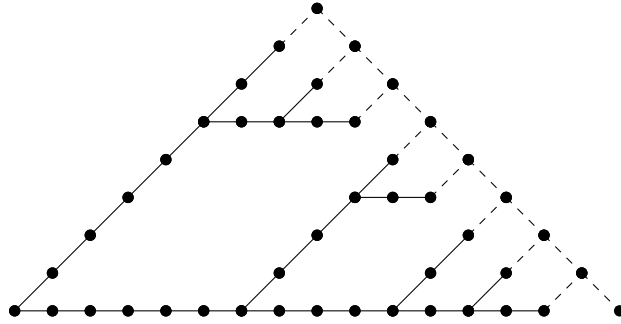
$$S_5^5 = ((\varepsilon, (\varepsilon, \varepsilon)), ((\varepsilon, \varepsilon), (\varepsilon, (\varepsilon, \varepsilon))))$$

$$\text{procp } S_5^5 = \varphi_0 + \varphi_6 + \varphi_7 - \varphi_{13} - \varphi_{15} - \varphi_{16}$$

$$\text{resp } S_5^5 = 2\varphi_0 + \varphi_5 + \varphi_6 + \varphi_7 - \varphi_{11} - \varphi_{13} - \varphi_{14} - \varphi_{15} - \varphi_{16}$$

$$\ell(S_5^5) = 8$$
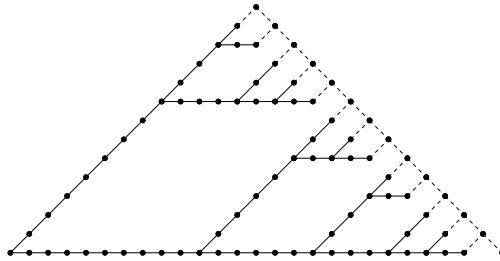
$$S_6^6 = (((\varepsilon, \varepsilon), (\varepsilon, (\varepsilon, \varepsilon))), ((\varepsilon, (\varepsilon, \varepsilon)), ((\varepsilon, \varepsilon), (\varepsilon, (\varepsilon, \varepsilon)))))$$

$$\text{procp } S_6^6 = \varphi_0 + \varphi_{10} + \varphi_{12} + \varphi_{13} - \varphi_{15} + \varphi_{16} - \varphi_{18} - \varphi_{23} - \varphi_{25} - \varphi_{26}$$

$$\text{resp } S_6^6 = 2\varphi_0 + \varphi_8 + \varphi_{10} + \varphi_{11} + \varphi_{12} - \varphi_{18} - \varphi_{21} - \varphi_{23} - \varphi_{24} - \varphi_{25} - \varphi_{26}$$

$$\ell(S_6^6) = 13$$

Figure 10.: Optimal schedules for $p = k$. These Fibonacci schedules are also optimal for $\left\lceil \frac{k+1}{2} \right\rceil \leq p \leq k$.

4.2. Exhaustive Search

An exhaustive search has been implemented to find optimal schedules for small values p and k. This search has been implemented in OCaml, more to that in Appendix A. One particularly interesting schedule discovered by the search is presented in Appendix D.

The search result is summarized in Figure 11, which lists the maximum reach $\ell_{\max}(p, k)$ of the found schedules. Unknown values that could not be computed within an acceptable amount of time are marked as "?". Rows and columns with known continuation are marked as "$\cdots$" and will be explained in the following sections. Columns whose continuation is conjectured are marked as "$(\cdots)$". Values which are inapplicable because of $p > k$ are marked as "—". Values which correspond to the already known Fibonacci schedules ($p \geq \lceil \frac{k+1}{2} \rceil$) are written in parentheses.

To increase the range of the exhaustive search, it has been reformulated via integer linear programming and also via Petri nets. [Che14, p. 9] Interval Petri nets [PZ13] have been considered, too, but were discarded because they are even more computationally expensive to analyze, and because classic Petri nets are already sufficient to describe parallel reversal schedules. Unfortunately, none of these approaches provided any improvement over the directly implemented search.

Moreover, the need for an exhaustive search diminished, given the schedules presented in the following sections.

$\ell_{\max}(p, k)$	$p = 1$	$p = 2$	$p = 3$	$p = 4$	$p = 5$	$p = 6$	$p \geq 7$
$k = 1$	(1)	—	—	—	—	—	—
$k = 2$	1	(2)	—	—	—	—	—
$k = 3$	1	(3)	(3)	—	—	—	—
$k = 4$	1	4	(5)	(5)	—	—	—
$k = 5$	1	5	(8)	(8)	(8)	—	—
$k = 6$	1	6	12	(13)	(13)	(13)	—
$k = 7$	1	7	17	(21)	(21)	(21)	$\cdots$
$k = 8$	1	8	23	32	(34)	(34)	$\cdots$
$k = 9$	1	9	30	47	(55)	(55)	$\cdots$
$k = 10$	1	10	38	67	84	(89)	$\cdots$
$k = 11$	1	11	47	?	?	(144)	$\cdots$
$k = 12$	1	12	57	?	?	?	
$k \geq 13$	$\cdots$	$\cdots$	$(\cdots)$				

Figure 11.: Maximum reach $\ell_{\max}$ of small schedules. Notation: "?" = unknown, "$\cdots$" = continuation is known, "$(\cdots)$" = continuation is conjectured, "—" = inapplicable, "(F_n)" = corresponds to a Fibonacci schedule.

4.3. One Process

Definition 72. For $p = 1$, we define:

$$S_k^1 := \varepsilon \quad \text{for all } k \geq 2$$

Note that $S_1^1 = \varepsilon$ is already defined in Section 4.1.

Example 73. For completeness, this simple schedule is depicted in Figure 12.

Theorem 74. *The schedules S_k^1 are optimal, that is:*

$$\ell_{\max}(1, k) = \ell(S_k^1) = 1$$

The profiles are:

$$\text{procp } S_k^1 = \varphi_0 - \varphi_2 \qquad\qquad \text{proc } S_k^1 = 1$$
$$\text{resp } S_k^1 = \varphi_0 - \varphi_2 \qquad\qquad \text{res } S_k^1 = 1$$

Proof. $S_1^1 = \varepsilon$ is the only schedule that needs just one process, because all schedules for reversal of more than one physical step need to run a P and an R action in parallel. Since this is the only schedule for $p = 1$, it is the optimum. The profiles are given by Lemma 63. The maximum resources and processes follow directly from the profiles. $\square$

$$S_1^1 = S_2^1 = \cdots = S_k^1 = \varepsilon$$

Figure 12.: Optimal schedules (all identical) for $p = 1$.

4.4. Two Processes

Definition 75. For $p = 2$, we define:

$$S_k^2 := \left(S_{k-1}^2, \varepsilon \right) \quad \text{for } k \geq 3$$

Note that $S_2^2 = (\varepsilon, \varepsilon)$ is already defined in Section 4.1.

Example 76. Some of these schedules are shown in Figure 13.

Theorem 77. *The schedules S_k^2 for $k \geq 3$ are optimal, that is:*

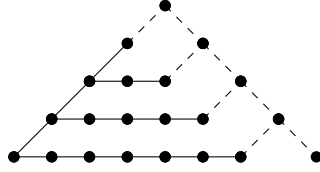$$\ell_{\max}(2, k) = \ell(S_k^2) = k$$

The profiles are:

$$\text{procp } S_k^2 = \varphi_0 + \varphi_k - \varphi_{2k-1} - \varphi_{2k} \qquad\qquad \text{proc } S_k^2 = 2$$

$$\text{resp } S_k^2 = 2\varphi_0 + \sum_{i=1}^{k-2} \varphi_i - \sum_{i=k+1}^{2k} \varphi_i \qquad\qquad \text{res } S_k^2 = k$$

Proof. The profiles are calculated via Lemma 64. The reach $\ell(S_k^2) = k$ follows directly from the profile durations $\left| \text{procp } S_k^2 \right| = \left| \text{resp } S_k^2 \right| = 2k$. The maximum resources and processes also follow directly from the profiles, as the maximal partial sum of the coefficients.

We will show optimality by contradiction. Assume for some k that we found a schedule S' which needs at most k resources, but reverses $\ell = k + 1$ physical steps. During reversal, the two processes are needed for the parallel P and R actions, so there can't be any F action during the second half of the schedule. This means that all physical steps $k - 1, \ldots, 1$ must be given by C actions. In particular, during the first reversal we have:

- one R action for the physical step $k + 1$,
- one P action for the physical step k, and
- one C action for each of the physical steps $1, \ldots, k - 1$.

In total, we have $k + 1$ actions in parallel, which contradicts our assumption that S' needs at most k resources. Hence, the S_k^2 are optimal. □

$$S_4^2 = (((\varepsilon, \varepsilon), \varepsilon), \varepsilon)$$
$$\text{procp}\, S_4^2 = \varphi_0 + \varphi_4 - \varphi_7 - \varphi_8$$
$$\text{resp}\, S_4^2 = 2\varphi_0 + \varphi_1 + \varphi_2 - \varphi_5 - \varphi_6 - \varphi_7 - \varphi_8$$
$$\ell(S_4^2) = 4$$

$$S_5^2 = ((((\varepsilon, \varepsilon), \varepsilon), \varepsilon), \varepsilon)$$
$$\text{procp}\, S_5^2 = \varphi_0 + \varphi_5 - \varphi_9 - \varphi_{10}$$
$$\text{resp}\, S_5^2 = 2\varphi_0 + \varphi_1 + \varphi_2 + \varphi_3 - \varphi_6 - \varphi_7 - \varphi_8 - \varphi_9 - \varphi_{10}$$
$$\ell(S_5^2) = 5$$

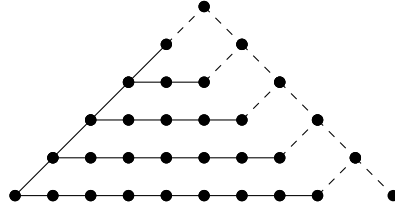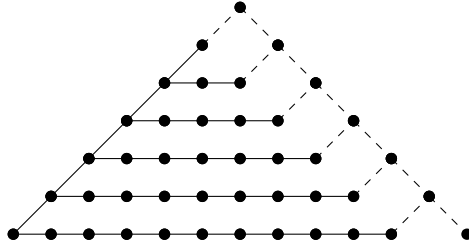$$S_6^2 = (((((\varepsilon, \varepsilon), \varepsilon), \varepsilon), \varepsilon), \varepsilon)$$
$$\text{procp}\, S_6^2 = \varphi_0 + \varphi_6 - \varphi_{11} - \varphi_{12}$$
$$\text{resp}\, S_6^2 = 2\varphi_0 + \varphi_1 + \varphi_2 + \varphi_3 + \varphi_4 - \varphi_7 - \varphi_8 - \varphi_9 - \varphi_{10} - \varphi_{11} - \varphi_{12}$$
$$\ell(S_6^2) = 6$$

Figure 13.: Optimal schedules for $p = 2$.

4.5. Three Processes

Definition 78. For $p = 3$, we define:

$$S_4^3 := \left(S_3^2, S_2^2\right)$$
$$S_k^3 := \left(S_{k-1}^3, S_{k-2}^2\right) \quad \text{for } k \geq 5$$

Example 79. Some of these schedules are shown in Figure 14.

Theorem 80. *The profiles of S_k^3 for $k \geq 4$ are:*

$$\text{procp } S_k^3 = \varphi_0 + \varphi_\ell + \varphi_{\ell+1} - \varphi_{2\ell-k+1} - \varphi_{2\ell-1} - \varphi_{2\ell} \qquad \text{proc } S_k^3 = 3$$

$$\text{resp } S_k^3 = 2\varphi_0 + \sum_{i=1}^{k-2} \varphi_{m_i} - \sum_{i=2\ell-k+1}^{2\ell} \varphi_i \qquad \text{res } S_k^3 = k$$

where:

$$\ell = 2 + \frac{(k-2)(k-1)}{2}$$
$$m_i = ki - \frac{i(i-3)}{2}$$

Their reach is:

$$\ell(S_k^3) = 2 + \frac{(k-2)(k-1)}{2}$$

Proof. The profiles are calculated via Lemma 64. The reach, maximum resources and maximum processes follow directly from the profiles. $\qquad\square$

Remark. Note that ℓ and m_i can also be written as:

$$\ell = 2 + \sum_{j=1}^{k-2} j$$

$$m_i = \sum_{j=k-i-1}^{k-2} j$$

which are close to the schedule structures as depicted in Figure 14.

Motivation. The exhausive search in Figure 11 shows that all schedules S_k^3 have exactly the optimal reach for $4 \leq k \leq 12$. It is likely that these are optimal for all k.

Conjecture 81. *The schedules S_k^3 are optimal, that is:*

$$\ell_{\max}(3, k) = \ell(S_k^3) = 2 + \frac{(k-2)(k-1)}{2}$$

$$S_4^3 = (((\varepsilon, \varepsilon), \varepsilon), (\varepsilon, \varepsilon))$$
$$\text{procp } S_4^3 = \varphi_0 + \varphi_5 + \varphi_6 - \varphi_7 - \varphi_9 - \varphi_{10}$$
$$\text{resp } S_4^3 = 2\varphi_0 + \varphi_2 + \varphi_3 - \varphi_7 - \varphi_8 - \varphi_9 - \varphi_{10}$$
$$\ell(S_4^3) = 5$$

$$S_5^3 = ((((\varepsilon, \varepsilon), \varepsilon), (\varepsilon, \varepsilon)), ((\varepsilon, \varepsilon), \varepsilon))$$
$$\text{procp } S_5^3 = \varphi_0 + \varphi_8 + \varphi_9 - \varphi_{12} - \varphi_{15} - \varphi_{16}$$
$$\text{resp } S_5^3 = 2\varphi_0 + \varphi_3 + \varphi_5 + \varphi_6 - \varphi_{12} - \varphi_{13} - \varphi_{14} - \varphi_{15} - \varphi_{16}$$
$$\ell(S_5^3) = 8$$

$$S_6^3 = (((((\varepsilon, \varepsilon), \varepsilon), (\varepsilon, \varepsilon)), ((\varepsilon, \varepsilon), \varepsilon)), (((\varepsilon, \varepsilon), \varepsilon), \varepsilon))$$
$$\text{procp } S_6^3 = \varphi_0 + \varphi_{12} + \varphi_{13} - \varphi_{19} - \varphi_{23} - \varphi_{24}$$
$$\text{resp } S_6^3 = 2\varphi_0 + \varphi_4 + \varphi_7 + \varphi_9 + \varphi_{10} - \varphi_{19} - \varphi_{20} - \varphi_{21} - \varphi_{22} - \varphi_{23} - \varphi_{24}$$
$$\ell(S_6^3) = 12$$

Figure 14.: Optimal schedules for $p = 3$.

4.6. Four or more Processes

For $p = 1, 2, 3$ we observed a constant, linear and quadratic pattern in the schedule reaches $\ell(S_k^p)$ as well as in the schedules themselves. Unfortunately, this does not extend to $p \geq 4$. Here we do not observe schedules of cubic growth, they are smaller. More precisely, from the exhaustive search as of Figure 11 we know that the values of $\ell_{\max}(4, k)$ for $k = 4, 5, 6, 7, 8$ are $\ell = 5, 8, 13, 21, 32$. Since a cubic function is uniquely determined by 4 values, we know that the sequence $5, 8, 13, 21, \ldots$ would have to continue with 33, but it continues with 32. In other words, for $p \geq 4$ the schedules do not reach as far as we might expect.

Since the optimal schedules for $p \geq 4$ could not yet be found in a systematic way, we provide suboptimal schedules. These grow quadratic with k and cubic with p. To construct them, we start with defining the subschedules to be used.

Definition 82 (Total recalculation schedules). For $\ell \geq 1$, we define T_ℓ to be the total recalculation schedules of reach ℓ:

$$T_1 := \varepsilon$$
$$T_\ell := (\varepsilon, T_{\ell-1}) \quad \text{for } \ell \geq 2$$

Definition 83 (Piled total recalculation schedules). For $p \geq 2$ and $k \geq p$, we define P_k^p to be the composition of uniform total recalculation schedules to form a pile, with a smaller schedule T_2 on the top:

$$P_p^p := T_2$$
$$P_k^p := \left(P_{k-1}^p, T_{2p-3}\right) \quad \text{for } k \geq p + 1$$

Lemma 84. *For all schedules P_k^p with $p \geq 3$ and $k \geq p + 1$:*

$$\ell(P_k^p) = 2 + (k-p)(2p-3) \qquad \operatorname{proc} P_k^p = p \qquad \operatorname{res} P_k^p = k$$

$$\operatorname{procp} P_k^p = \varphi_0 + \sum_{i=1}^{p-2} \varphi_{m+2+2i} + \varphi_\ell - \sum_{i=1}^{p-1} \varphi_{\ell+m+2i} - \varphi_{2\ell}$$

$$\operatorname{resp} P_k^p = 2\varphi_0 + \sum_{i=1}^{k-p-1} \varphi_{i(2p-3)} + \sum_{i=1}^{p-3} \varphi_{m+2+2i} + \varphi_{\ell-2} + \varphi_{\ell-1}$$
$$- \sum_{i=1}^{k-p-1} \varphi_{\ell+i(2p-3)} - \sum_{i=1}^{p-2} \varphi_{\ell+m+2i} - \varphi_{2\ell-2} - \varphi_{2\ell-1} - \varphi_{2\ell}$$

where:

$$\ell = \ell(P_k^p)$$
$$m = \ell - 2p + 1$$

Definition 85 (Suboptimal schedules). For $p \geq 4$, we define:

$$S_{p+1}^p := \left(P_p^{p-1}, T_{2p-5}\right)$$
$$S_k^p := \left(S_{k-1}^p, P_{k-2}^{p-1}\right) \quad \text{for } k \geq p+2$$

Example 86. Figure 15 demonstrates the construction of S_8^4 and shows some of the used subschedules.

Theorem 87. *The schedules S_k^p for $p \geq 4$ and $k \geq p+1$ have the following reaches:*

$$\ell(S_k^p) = \left(\frac{(k-p)(k-p-1)}{2} + 2\right)(2p-5) + 2(k-p)$$

For $k \geq p+2$ they have the following processes profile:

$$\text{procp } S_k^p = \varphi_0 + \sum_{i=1}^{p-3} \varphi_{m+4+2i} + \varphi_\ell + \varphi_{\ell+2p-5} - \varphi_n - \sum_{i=1}^{p-2} \varphi_{\ell+m+2+2i} - \varphi_{2\ell}$$
$$\text{proc } S_k^p = p$$

where:

$$\ell = \ell(P_k^p)$$
$$m = \ell - 2p + 1$$
$$n = 2\ell - (2p-5)(k-p) - 2$$

Proof. The profile is calculated via Lemma 64 using Lemma 84. The reach and maximum processes follow directly from the profile. $\square$

Conjecture 88. *For $k \geq p+2$ the suboptimal schedules all satisfy the resources requirement:*
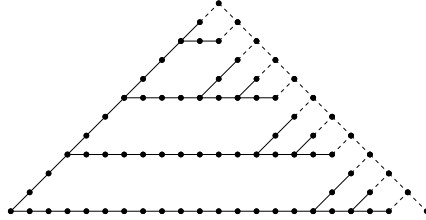
$$\text{res } S_k^p = k$$
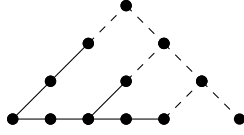
$$T_3 = (\varepsilon, (\varepsilon, \varepsilon))$$
$$\operatorname{procp} T_3 = \varphi_0 + \varphi_2 - \varphi_5 - \varphi_6$$
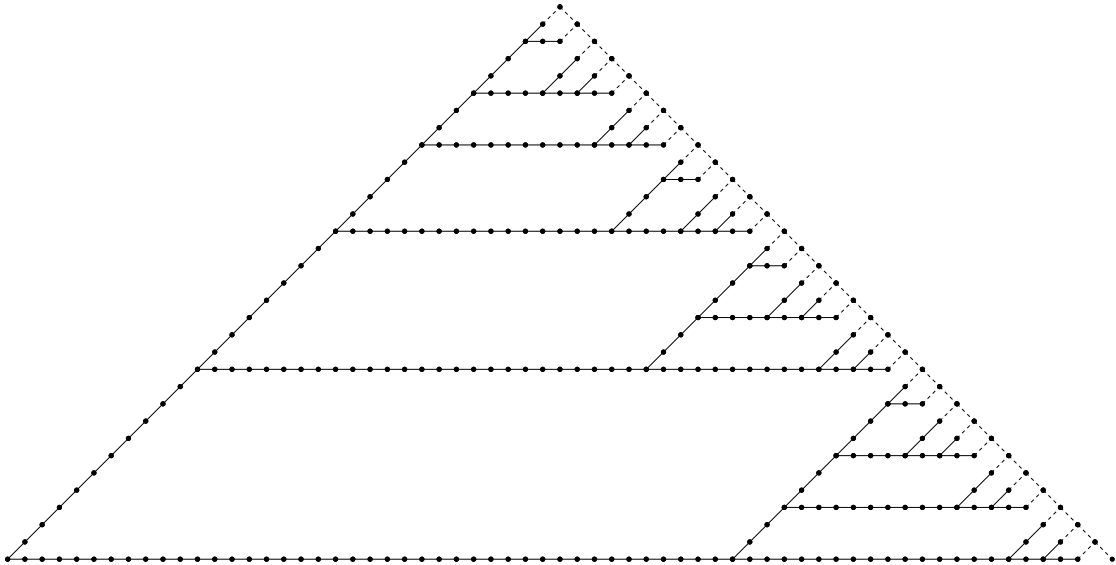$$\operatorname{resp} T_3 = 2\varphi_0 + \varphi_2 - \varphi_4 - \varphi_5 - \varphi_6$$

$$P_6^3 = (((T_2, T_3), T_3), T_3)$$
$$\operatorname{procp} P_6^3 = \varphi_0 + \varphi_{10} + \varphi_{11} - \varphi_{19} - \varphi_{21} - \varphi_{22}$$
$$\operatorname{resp} P_6^3 = 2\varphi_0 + \varphi_3 + \varphi_6 + \varphi_9 + \varphi_{10} - \varphi_{14} - \varphi_{17} - \varphi_{19} - \varphi_{20} - \varphi_{21} - \varphi_{22}$$

$$S_8^4 = ((((P_4^3, T_3), P_4^3), P_5^3), P_6^3)$$
$$\operatorname{procp} S_8^4 = \varphi_0 + \varphi_{31} + \varphi_{32} + \varphi_{35} - \varphi_{50} - \varphi_{61} - \varphi_{63} - \varphi_{64}$$
$$\operatorname{resp} S_8^4 = 2\varphi_0 + \varphi_{11} + \varphi_{19} + \varphi_{24} + \varphi_{27} + \varphi_{30} + \varphi_{31}$$
$$- \varphi_{50} - \varphi_{53} - \varphi_{56} - \varphi_{59} - \varphi_{61} - \varphi_{62} - \varphi_{63} - \varphi_{64}$$
$$\ell(S_8^4) = 32$$

Figure 15.: Construction of schedule S_8^4.

4.7. Summary

Figure 16 shows the reach of the optimal and suboptimal schedules defined so far. For comparison, for each new schedule the reach of the largest applicable Fibonacci schedule is shown in parentheses. For small k the Fibonacci schedules are usually better. For larger k the advantages of the new schedules become apparent.

Comparing the reach of the suboptimal schedules ($p \geq 4$) with the optima calculated by exhaustive search (Figure 11), we see that S_8^4 is indeed optimal. Moreover, S_9^4 with reach $46 < 47$ and S_{10}^4 with reach $63 < 67$ are close to optimal. However, S_{10}^5 with reach $70 < 84$ indicates that with larger p and k the gap will probably become larger.

$\ell\left(S_k^p\right)$	$p = 2$	$p = 3$	$p = 4$	$p = 5$	$p = 6$
$k = 1$	—	—	—	—	—
$k = 2$	2 (3)	—	—	—	—
$k = 3$	3 (3)	3 (3)	—	—	—
$k = 4$	4 (3)	5 (5)	5 (5)	—	—
$k = 5$	5 (3)	8 (8)	8 (8)	8 (8)	—
$k = 6$	6 (3)	12 (8)	13 (13)	12 (13)	13 (13)
$k = 7$	7 (3)	17 (8)	21 (21)	19 (21)	16 (21)
$k = 8$	8 (3)	23 (8)	32 (21)	31 (34)	25 (34)
$k = 9$	9 (3)	30 (8)	46 (21)	48 (55)	41 (55)
$k = 10$	10 (3)	38 (8)	63 (21)	70 (55)	64 (89)
$k = 11$	11 (3)	47 (8)	83 (21)	97 (55)	94 (144)
$k = 12$	12 (3)	57 (8)	106 (21)	129 (55)	131 (144)
$k = 20$	20 (3)	173 (8)	398 (21)	565 (55)	679 (144)
$k = 30$	30 (3)	408 (8)	1033 (21)	1560 (55)	1994 (144)
$k = 50$	50 (3)	1178 (8)	3203 (21)	5050 (55)	6724 (144)
$k = 100$	100 (3)	4853 (8)	13878 (21)	22525 (55)	30799 (144)
$k = 200$	200 (3)	19703 (8)	57728 (21)	94975 (55)	131449 (144)

Figure 16.: Reach $\ell\left(S_k^p\right)$ of all schedules defined so far. The numbers in parentheses show the reach of the largest applicable Fibonacci schedule.

5. Conclusion

As expected, relaxing the central assumption of processor-checkpoint convertibility was very challenging, and while the research presented within this thesis cannot be a comprehensive discussion on that topic, it does provide new insights into what the new schedules look like, as well as new approaches and tools to analyze them.

Using a new and more compact representation of schedules, it was shown that checkpoint persistence still holds, while processor persistence and hence binary decomposition do not hold anymore. A comprehensive algebra on profile functions was developed, which provided a compact description of how the schedule profiles behave under various operations. This algebra enabled the analysis of associators of profiles, which in turn led to inequalities that ruled out a whole class of schedule compositions that are guaranteed to provide non-optimal schedules. The profile algebra turned out to be an appropriate instrument to analyze parallel reversal schedule via their profiles. New optimal schedules for a small numbers of processes were developed, to be applied in situations where the known Fibonacci schedules are not applicable. Additionally, suboptimal schedules were created where optimal schedules could not be found in a systematic way.

Future research may try to find the remaining optimal schedules, or may try to improve the suboptimal schedules presented here. Also, more assumptions should be relaxed to make the schedules more realistic. Relaxing the processor-checkpoint convertibility is only one of many possibilities. For example, the assumption $t = \hat{t} = \bar{t} = 1$ could be relaxed, to take into account that P and R actions usually take more time than F actions, which has been done for the Fibonacci schedules but was outside the scope of the new schedules. In addition, all schedules so far ignore communication costs of transferring data within a parallel computer system, which is also an open issue for the Fibonacci schedules.

The profile algebra may be useful in other research fields, as it is very generic at its core. It should be possible to apply this algebra to any sequence of events, where an *event* is considered to by anything that affects resource usage at a certain point in time. Moreover, the algebra could be generalized to handle multiple resources at once. For example, instead of having separate profiles for processes and resources, there could be combined profiles that describe processes and resources at once. This generalized algebra would be defined on functions $f\colon \mathbb{R} \to \mathbb{R}^n$ rather than $f\colon \mathbb{R} \to \mathbb{R}$, where a typical profile might look like this: $\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}\varphi_0 + \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}\varphi_2 + \begin{pmatrix} -1 \\ -1 \\ 0 \end{pmatrix}\varphi_3$. The partial order $\leq$ would generalize in the usual way and almost all rules of Section 3.5 should still apply. Only the rules involving φ_t would have to be rewritten using the new primitive functions $\mathcal{B} = \{e_i\varphi_t \mid i \in \{1,\ldots,n\}, t \in \mathbb{R}_{\geq 0}\}$.

A. Tool for Generating Schedules

Along with this thesis a program has been developed to support the research. Most of what is formally described here, such as the profile algebra, is also implemented within that tool. Almost all schedule pictures and profiles have been automatically generated by this tool. Also, the exhaustive search as of Section 4.2 is implemented within that tool.

It is implemented in OCaml [INR15], a programming language that provides very good support for high-level programming, while compiling down to efficient native code whose speed to comparable to C and C++. The correctness of the program is secured by the strict ML typesystem combined with a comprehensive set of unit tests. This allows for fast refactoring, which enabled the program to be developed simultaneously with the theory, influencing each other.

The tool and related resources will be published at:

<div align="center">

`https://njh.eu/prs`

</div>

If the tool "`prs`" is called without any arguments, the list of commands is shown:

```
Usage:
prs s PROCESSES RESOURCES
prs sp PROCESSES RESOURCES
prs gen PROCESSES RESOURCES
prs tree TREE
prs tree_letters TREE
prs tree_search MAX_PROCESSES MAX_RESOURCES MAX_REACH
prs old_tree_search PROCESSES MAX_RESOURCES
prs ascii < PRIME_SCHEDULE
prs ascii_small < PRIME_SCHEDULE
prs fibers FIBERS
prs lp PROCESSES REACH
prs search MAX_PROCESSES MAX_RESOURCES
prs latex_tree TREE
prs latex_gen PROCESSES RESOURCES
prs latex_ps < PRIME_SCHEDULE
prs test
```

The command "`prs test`" runs all unit tests. "`prs search`" runs the exhaustive search. "`prs gen`" generates the optimal respectively suboptimal schedules as defined in Chapter 4. The commands "`prs s`" and "`prs sp`" also generate those schedules, but have

a less verbose output. Arbitrary binary-decomposable schedules can be analyzed with "prs tree".

For example, "prs gen 3 6" generates the optimal schedule for $p = 3, k = 6$, draws it in ASCII art, shows the tree structure (where e means ε) and shows the profiles (where -1@23 means $-\varphi_{23}$). Note that the ASCII art schedules are rotated by $-90°$.

```
1   2   |\
1   2   |.\
1   2   |..\
1   2   |...\
1   3   |...|\
1   3   |...|.\
1   3   |...|..\
1   4   |...|..|\
1   4   |...|..|.\
1   5   |...|..|.|\
1   6   |...|..|.||\
1   6   |...|..|.||.\
2   6   |...|..|.|.\/
3   6   |...|..|\.\/
3   6   |...|\.|.\/
3   6   |...||\.\/
3   6   |\..||.\/
3   6   ||\.|.\/
3   6   |||\.\/
2   5   |||.\/
2   4   ||.\/
2   3   |.\/
2   2   .\/
1   1   ./

3   6   12

S     = (((((e,e),e),(e,e)),((e,e),e)),(((e,e),e),e))
procp = +1@0,+1@12,+1@13,-1@19,-1@23,-1@24
resp  = +2@0,+1@4,+1@7,+1@9,+1@10,-1@19,-1@20,-1@21,-1@22,-1@23,-1@24
reach = 12
```

B. Time Split in Schedule Algebra

This appendix demonstrates how to determine what exactly happens within a schedule f at a certain point in time t, when f is decribed by the schedule algebra as of Chapter 2, To answer that question, we introduce a *time split* operation that splits f into a subschedule before t and a list of subschedules starting at t.

Note that this approach is mentioned just for reference. The recommended way to deal with time is through schedule profiles as explained in Chapter 3.

Definition 89 (Shortest task duration)**.** The *shortest task duration $o(f)$* of a schedule $f \in \mathbb{Z}\langle A \rangle$ is the duration of its shortest task:[1]

$$o(f) = \begin{cases} 0 & \text{for } f = 0 \\ \min\left\{|w_1|, \ldots, |w_n|\right\} & \text{for } f = a_1 w_1 + \cdots + a_n w_n, w_i \in A^*, a_i \in \mathbb{Z} \setminus \{0\} \end{cases}$$

Lemma 90 (Time split)**.** *For every schedule $f \in \mathbb{Z}\langle A \rangle$ and time $t \in \{1, \ldots, o(f)\}$ there exist tasks $w_1, \ldots, w_n \in A^*$ and schedules $g_1, \ldots, g_n \in \mathbb{Z}\langle A \rangle$ such that*

$$f = w_1 g_1 + \cdots + w_n g_n = \begin{pmatrix} w_1 & \cdots & w_n \end{pmatrix} \begin{pmatrix} g_1 \\ \cdots \\ g_n \end{pmatrix} \tag{43}$$

where $|w_i| = t$ and the w_i are pairwise distinct and uniquely determined except for their order.

Proof. From the preconditions it follows that $f \neq 0$, because for $f = 0$ there is no $t \in \{1, \ldots, o(0)\} = \varnothing$. So we can represent f as:

$$f = a_1 u_1 + \cdots + a_k u_k \quad \text{with } u_j \in A^*, a_j \in \mathbb{Z} \setminus \{0\}$$

For every u_j $(j = 1, \ldots, k)$ we have $t \leq o(f) \leq |u_j|$, hence we can split u_j into a prefix p_j of duration t and a suffix s_j that may be empty:

$$u_j = p_j s_j \quad \text{with } p_j, s_j \in A^*, |p_j| = t$$

Let $T = \{p_1, \ldots, p_k\}$ be the set of prefixes and $n = |T|$. Let $\{w_1, \ldots, w_n\} = T$ be a representation of T with pairwise distinct w_i for $i = 1, \ldots, n$. Then every p_j is equal to exactly one of the w_i. For each $i = 1, \ldots, n$ we define g_i to contain all suffixes (with

---

[1]This is also known as the *order* [Coh85, p. 60], but that term would be confusing as soon as when we use it in its other meaning to describe the order within a sequence.

their coefficients) of tasks whose prefix is p_i:

$$g_i = \sum_{\substack{j \\ p_j = w_i}} a_j s_j$$

This satisfies (43):

$$\sum_{i=1}^{n} w_i g_i = \sum_{i=1}^{n} \sum_{\substack{j \\ p_j = w_i}} a_j w_i s_j = \sum_{i=1}^{n} \sum_{\substack{j \\ p_j = w_i}} a_j p_j s_j = \sum_{i=1}^{n} \sum_{\substack{j \\ p_j = w_i}} a_j u_j = \sum_{j=1}^{k} a_j u_j = f$$
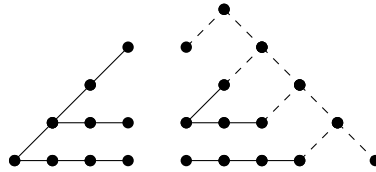
The uniqueness follows from the fact that all w_i have the same duration, so these must be exactly the prefixes of the tasks of f, and those prefixes are uniquely determined except for their order. $\qquad\square$

Example 91. We consider again the schedule of Example 12:

$$f = F^3 PR + FC^2 FPR + FC^4 PR + C^6 PR$$

Splitting f at $t = 3$ yields:

$$f = \begin{pmatrix} F^3 & FC^2 & C^3 \end{pmatrix} \begin{pmatrix} PR \\ FPR + C^2 PR \\ C^3 PR \end{pmatrix}$$

Remark. This lemma can be generalized to work for all $t \in \{1, \ldots, |f|\}$ rather than $t \in \{1, \ldots, o(f)\}$, by introducing an additional summand that contains all tasks whose duration is small than t.

C. Polynomials as Profiles

Motivation. The profile operations defined in Chapter 3 generalize various well-known operations on polynomials:

1. The *addition* and *scalar multiplication* translate directly to polynomials.
2. The *shift* operation $\gg$ generalizes the multiplication with monomials $X^a \cdot P$.
3. The *duration* operation $|f|$ generalizes the degree of polynomials: $\deg P$.
4. The *final value* $f(|f|)$ generalizes the evaluation at $X = 1$, which is also known as the sum of coefficients.

While the connection between polynomials and profile functions became obvious early in the work on this thesis, it is of no apparent use throughout. The main problems are:

1. Polynomial multiplication doesn't make sense on profiles. While it does make sense in the special cases of multiplication with scalars $v \cdot P$ and multiplication with monomials $X^a \cdot P$, the multiplication of any two profiles has no apparent meaning.
2. The partial order $\leq$, an essential part of the algebra, doesn't translate well to polynomials. For example, we might hope that nonnegativity could be handled by treating polynomials as functions. However, consider the following polynomial and its corresponding profile:

$$P = X^0 - 2X^1 + X^2 = (1 - X)^2$$
$$f = \varphi_0 - 2\varphi_1 + \varphi_2$$

   While P is a square and thus $P \geq 0$ by any natural definition, we have $f(t) = -1$ for all $t \in [1, 2)$ and hence $f \not\geq 0$. Of course, we can apply Lemma 47 directly to the coefficients of P, but then there disappears yet another advantage of using polynomials.

Nevertheless, it seems worth mentioning this connection, so we will describe it in exact terms.

Definition 92 (Polynomials as profiles)**.** We define Ψ to be the uniquely determined $\mathbb{R}$-linear map between the polynomials $\mathbb{R}[X]$ and the profiles $\mathbb{P}$ that maps their base elements as follows:

$$\Psi \colon \mathbb{R}[X] \to \mathbb{P}$$
$$\Psi \colon X^a \mapsto \varphi_a \quad (a \in \mathbb{Z}_{\geq 0})$$

Remark. Here we treat $\mathbb{R}[X]$ as an ordinary $\mathbb{R}$-linear space with basis $(X^0, X^1, X^2, \dots)$. That is, we ignore the possibility of polynomial multiplication and care only about scalar multiplication.

Lemma 93 (Polynomial operations on profiles). *Let* $F, G \in \mathbb{R}[X], v \in \mathbb{R}, a \in \mathbb{Z}_{\geq 0}$. *Further, let*

$$f = \Psi(F)$$
$$g = \Psi(G)$$

Then:

$$f + g = \Psi(F + G) \tag{44}$$
$$v \cdot f = \Psi(v \cdot F) \tag{45}$$
$$f \gg a = \Psi(X^a \cdot F) \tag{46}$$
$$|f| = \deg F \tag{47}$$
$$f(|f|) = F(1) \tag{48}$$

Proof. (44) and (45) follow directly from the definition of Ψ as an $\mathbb{R}$-linear map. The other equalities are obvious for $F = 0$. It remains to prove (46) to (48) for $F \neq 0$, which allows us to represent F as:

$$F = \sum_{i=0}^{n} a_i X^i \quad \text{with } a_n \neq 0$$

Then, we apply Ψ to both sides:

$$f = \sum_{i=0}^{n} a_i \varphi_i \quad \text{with } a_n \neq 0$$

Finally, we apply Lemma 44 at $t = n$ with $(t_1, \dots, t_{n+1}) = (0, \dots, n)$:

$$f(n) = \sum_{i=0}^{n} a_i$$

With all that in mind, we can prove (46) to (48):

$$f \gg a = \sum_{i=0}^{n} a_i \varphi_{i+a} = \sum_{i=0}^{n} a_i \Psi\left(X^{i+a}\right) = \Psi\left(\sum_{i=0}^{n} a_i X^{i+a}\right) = \Psi(X^a \cdot F)$$
$$|f| = n = \deg F$$
$$f(|f|) = f(n) = \sum_{i=0}^{n} a_i = F(1) \qquad \qquad \square$$

D. Notable Search Result for p=4, k=8

The exhaustive search produced an interesting optimal schedule for $p = 4$ and $k = 8$, which is shown in Figure 17. Here, processor persistence is violated and cannot be fixed locally. It is impossible to replace the violating C action with an F action, because at that time the full number of processes is running. It is only possible to fix this by replacing another F action at that time with a C action. But that introduces a violation at that new place, as demonstrated in Figure 18. Moving it further down even leads to two violations, as seen in Figure 19. Moreover, this second new schedule now uses $k = 9$ resources, hence it is no longer an optimal schedule. This example demonstrates that in general it is not possible to deal with violations of processor persistence through local modifications.

Note, however, that we know that this particular schedule can be fixed globally. In Section 4.6 we constructed the binary-composable (and therefore processor-persistent) schedule S_8^4, which has the same optimal reach $\ell = 32$.

Figure 17.: An optimal schedule for $p = 4, k = 8$ that contains a single violation of the processor persistence, marked as ●.
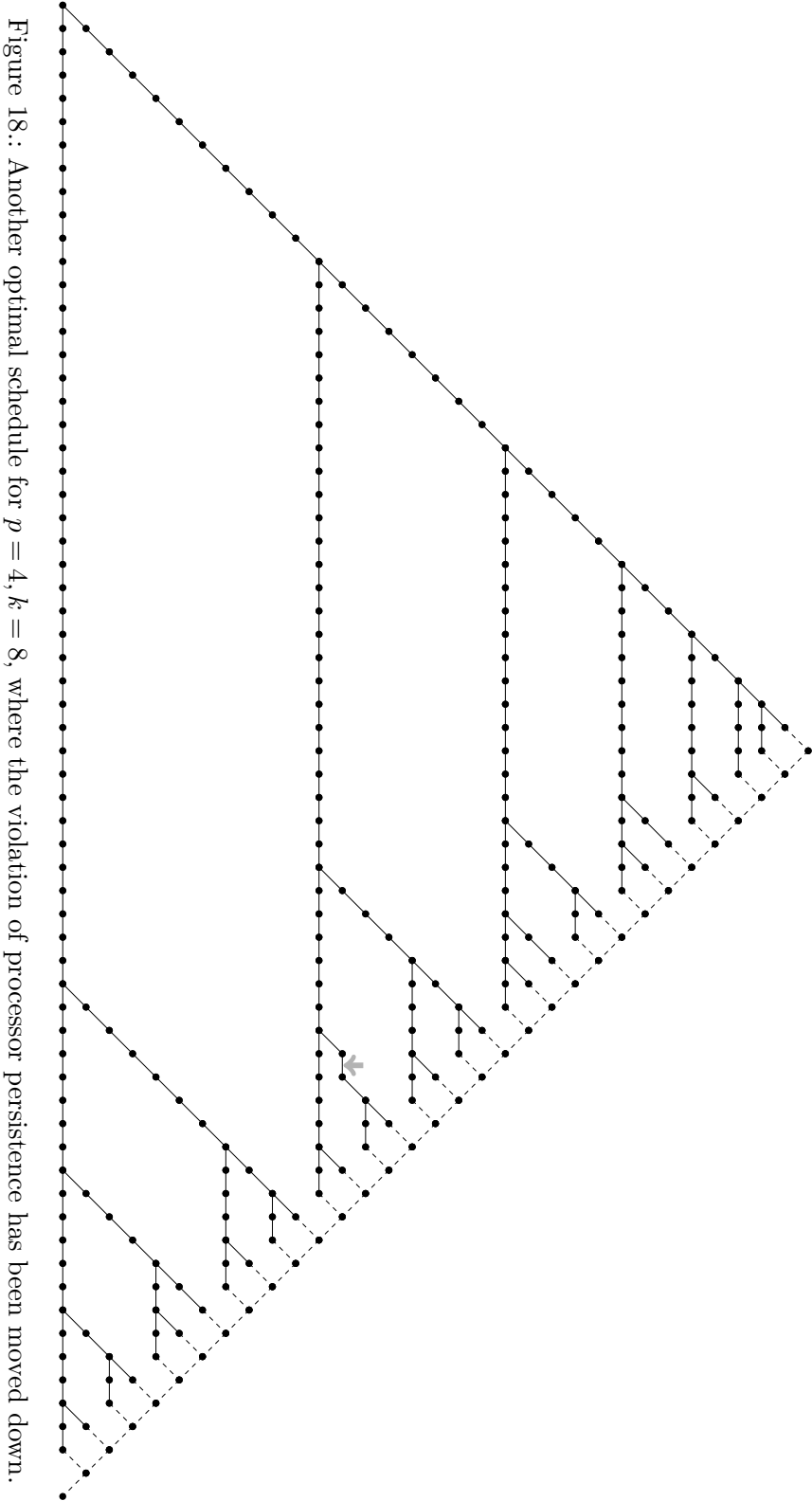
Figure 18.: Another optimal schedule for $p = 4, k = 8$, where the violation of processor persistence has been moved down.
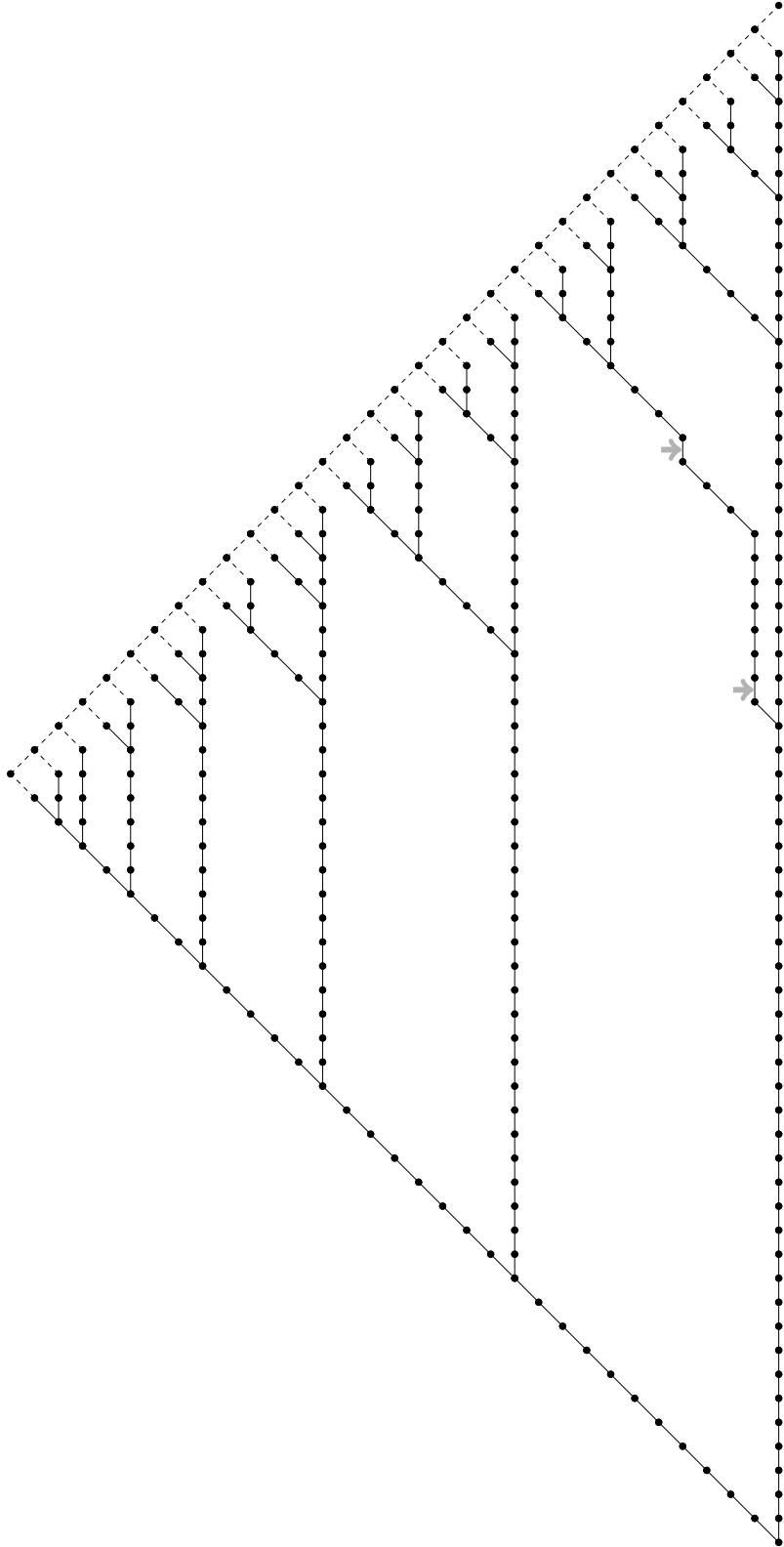
Figure 19.: Non-optimal schedule, created from Figure 18 by moving the violation of processor persistence further down. This is no longer an optimal schedule for $p = 4$, $k = 8$, as it now uses $k = 9$ resources.

List of Figures

Bibliography

[Ben73] BENNETT, Charles H.: Logical reversibility of computation. In: *IBM Journal of Research and Development* 17 (1973), Nr. 6, 525–532. `http://www.math.ucsd.edu/~sbuss/CourseWeb/Math268_2013W/Bennett_Reversibiity.pdf`. – SHA-256: `ad73fb54f0c0b97a3cddf0882598f84cc0e52cf16b7a851c214a854a38abce04`

[Bou87] BOURBAKI, Nicolas: *Elements of mathematics. Topological Vector Spaces, Chapters 1–5*. Springer, 1987 `http://d-nb.info/871297639`. – ISBN 978–3–540–13627–9

[Che14] CHEUNG, King S.: *Augmented marked graphs*. Springer, 2014 `http://dx.doi.org/10.1007/978-3-319-06428-4`. – ISBN 978–3–319–06428–4

[Coh85] COHN, Paul M.: *Free rings and their relations*. Second Edition. Academic Press, 1985 `http://www.maths.ed.ac.uk/~aar/papers/cohnfree.pdf`. – ISBN 978–0–12–179152–0. – SHA-256: `bdb4451b8ef1f5348958ce290669a0600d8adcdda5b9b4b368ce14d026d41d4f`

[GW00] GRIEWANK, Andreas ; WALTHER, Andrea: Algorithm 799: Revolve: An implementation of checkpointing for the reverse or adjoint mode of computational differentiation. In: *ACM Transactions on Mathematical Software* 26 (2000), 19–45. `http://www.researchgate.net/publication/220493433`. – SHA-256: `107af43db9b5a04ce81d3138af8302d643e5cfc02cd0afaffc67368ac86b65da`

[GW08] GRIEWANK, Andreas ; WALTHER, Andrea: *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Second Edition. Society for Industrial and Applied Mathematics, 2008 `http://epubs.siam.org/doi/book/10.1137/1.9780898717761`. – ISBN 978–0–89871–659–7

[INR15] INRIA – INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE: *Objective CAML (OCaml)*. `https://web.archive.org/web/20150914083528/https://ocaml.org/`. Version: 2015

[Jet86] JETER, Melvyn W.: *Mathematical programming – An introduction to optimization*. Dekker, 1986 `https://books.google.com/books?id=ofrBsl61lq8C`. – ISBN 978–0–8247–7478–3

[Leh03] LEHMANN, Uwe: *Schedules for dynamic bidirectional simulations on parallel computers*, Dresden University of Technology, Dissertation, 2003. `http://d-nb.info/968544541/34`. – SHA-256: `6fbd0f7668cdac2515045144ecc4dc761e4a34bdaadcb4ef1f56cb1e7cb6265d`

[LW02]   LEHMANN, Uwe ; WALTHER, Andrea: The Implementation and Test-
         ing of Time-minimal and Resource-optimal Parallel Reversal Schedules.
         Version: 2002.   http://dx.doi.org/10.1007/3-540-46080-2_110.   In:
         SLOOT, Peter M. A. (Hrsg.) ; HOEKSTRA, Alfons G. (Hrsg.) ; TAN, C. J. K.
         (Hrsg.) ; DONGARRA, Jack J. (Hrsg.): *Computational Science – ICCS 2002* Bd.
         2330. Springer Berlin Heidelberg, 2002. – ISBN 978–3–540–43593–8, 1049–1058

[NS12]   NAUMANN, Uwe ; SCHENK, Olaf: *Combinatorial Scientific Computing.*
         Springer, 2012 http://dx.doi.org/10.1201/b11644. –   ISBN 978–3–642–
         41115–1

[PZ13]   POPOVA-ZEUGMANN, Louchka: *Time and Petri Nets.* Springer, 2013 http:
         //dx.doi.org/10.1007/978-3-642-41115-1. – ISBN 978–3–642–41115–1

[Sch66] SCHAFER, Richard D.: *An introduction to nonassociative algebras.* Academic
         Press, 1966 https://books.google.com/books?id=CqgX7vtaCSMC. –   ISBN
         978–0–126–22450–4

[Sym07] SYMES, William W.:   Reverse time migration with optimal checkpointing.
         In: *Geophysics* 72 (2007), Nr. 5. http://www.caam.rice.edu/tech_reports/
         2006/TR06-18.pdf. –
         SHA-256: de4191ac55f082bcfd877ee39bcbd9ed8d6bb219ddfc392510b44aceb51693bf

[Wal99] WALTHER, Andrea: *Program reversal schedules for single and multi-processor
         machines*, Dresden University of Technology, Dissertation, 1999. http://d-nb.
         info/96395007X/34. –
         SHA-256: ec8902ad34d96536dee7e904737314c94539e3665752e00548a672955c442548

[Wal04] WALTHER, Andrea: Bounding the Number of Processors and Checkpoints
         Needed in Time-minimal Parallel Reversal Schedules. In: *Computing* 73 (2004),
         Nr. 2, 135–154. http://dx.doi.org/10.1007/s00607-004-0075-1

Statements

License / Lizenz
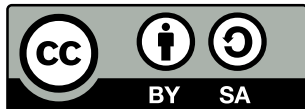
Selbständigkeitserklärung

Ich erkläre, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Berlin, den 24. September 2015

_____