

The Cost of Concurrent, Low-Contention Read-Modify-Write*

Costas Busch[†]

Marios Mavronicolas[‡]

Paul Spirakis[§]

Abstract

In this work, we embark on a study of the possibility (or impossibility), and the corresponding costs, of devising concurrent, low-contention implementations of atomic Read-Modify-Write operations (abbreviated as RMW), in a distributed system. We consider a natural class of RMW operations which give rise to a certain class of algebraic groups that we introduce here and call *monotone groups*. Our chief combinatorial instrument is a *Monotone Linearizability Lemma*, which establishes inherent ordering constraints of linearizability for a certain class of executions of *any* distributed system that implements a monotone RMW operation.

The end results of our study specifically apply to implementations of (monotone) RMW operations that are based on *switching networks*, a recently introduced class of concurrent, low-contention data structures that generalize *counting networks*. These results are negative and they are shown through a modular use of the Monotone Linearizability Lemma. In particular, we derive the *first* lower bounds on *size* (the number of *switches* in the network) and *latency* (the maximum number of switches traversed) for any (non-trivial) switching network implementing a monotone RMW operation:

- If the network is made up of switches with *finite* state and it incurs low contention, then it must contain an infinite number of switches, even if *concurrency* (the maximum number of concurrent processes) is restricted to remain bounded.
- If the network is made up of switches with *infinite* state and it incurs low contention, then it must still contain an infinite number of switches if we now allow concurrency to grow unbounded.
- Any switching network induces executions with latency at least $\left\lceil \frac{n-1}{c-1} \right\rceil$, where n is the number of concurrent processes and c is the maximum number of processes that simultaneously access a switch.

A major significance of the above lower bounds (of infinity) is that they formally explain the observed inability of researchers over the last decade to extend counting networks, while retaining them finite and low-contention, in order to perform tasks more complex than just incrementing a counter by one.

Keywords: Concurrency, Read-Modify-Write Operations, Linearizability, Switching Networks, Counting Networks.

*This work has been partially supported by the IST Program of the European Union under contract numbers IST-1999-14186 (ALCOM-FT) and IST-2001-33116 (FLAGS), by funds from the Joint Program of Scientific and Technological Collaboration between Greece and Cyprus, by the Greek General Secretariat for Research and Technology, and by research funds from Rensselaer Polytechnic Institute and University of Cyprus.

[†]**Contact Author.** Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY 12180. Part of the work of this author was performed while visiting Department of Computer Science, University of Cyprus. Email: buschc@cs.rpi.edu; Fax: 518-276-4033.

[‡]Department of Computer Science, University of Cyprus, 1678 Nicosia, Cyprus. Part of the work of this author was performed while visiting Department of Mathematics and Computer Science, University of Paderborn. Email: mavronic@ucy.ac.cy

[§]Department of Computer Engineering and Informatics, University of Patras, Rion, 265 00 Patras, Greece, & Computer Technology Institute, P. O. Box 1122, 261 10 Patras, Greece. Email: spirakis@cti.gr

1 Introduction

Motivation, Framework and Outline. A Read-Modify-Write shared variable [8, 13], henceforth abbreviated as RMW, is an abstract variable type that allows reading its old value, and determining (via some specific *operator*) and writing a new value back to it in a single, *atomic* (indivisible) RMW operation (cf. [15, Example 9.4.2]). A RMW operation is a strong synchronization primitive that allows for the design of efficient and transparent algorithms in the asynchronous shared memory model of distributed computation; see, e.g., the folklore algorithm for mutual exclusion described in [3, Section 4.3.2], or the scalable *ordered multicast* protocol of Herlihy *et al.* [11] that is based on a modular use of the distributed **Swap** operation, a special case of RMW. Due to their fundamental importance as synchronization primitives, it is most desirable to devise suitable distributed data structures for the construction of concurrent, low-contention implementations of RMW variables. Intuitively, the *contention* of an implementation measures the extent to which concurrent *processes* access the same memory location simultaneously; it has been argued quite convincingly that contention is a critical factor for the overall efficiency of shared memory algorithms (cf. [6]). The central question motivating this work is the possibility (or impossibility), and the corresponding incurred complexities, for concurrent, low-contention implementations of RMW shared variables.

We focus on a specific class of RMW operations whose associated operators give rise to a certain class of algebraic groups introduced and studied here, which we call *monotone groups*. A monotone group has a *total order* and a *monotone subdomain* associated with it; the latter enjoys a significant monotonicity property, which we call *monotonicity under composition*: applying the operator on an element from the monotone subdomain results to another element in the monotone subdomain that strictly dominates the initial one with respect to the total order. For example, the **Fetch&Add** operation (over the set of integers) clearly falls into the context of monotone groups, since adding a positive integer to a positive integer results in a larger positive integer; here, the monotone subdomain is the set of positive integers. So also does the **Fetch&Multiply** operation, and so on. A *monotone* RMW operation is one that is associated with a monotone group.

An abstract concept defined in relation to monotone groups is that of *n-wise independence*. Roughly speaking, n elements of a monotone group are *n-wise independent* if it is not possible to derive the identity element of the group through successive (specifically restricted though) applications of the operator on n of the elements or their inverses. A preliminary but significant property of monotone groups that we prove is that *every* monotone group is *n-wise independent*, in the sense of having *n-wise independent* elements. As we establish, the existence of *n-wise independent* elements in a monotone group is largely responsible for enforcing *linearizability* [12] for certain suitable executions of a distributed system that implements the corresponding (monotone) RMW operation; recall that an execution is *linearizable* [12] if the values returned to operations in it respect the real-time ordering of the operations.

As a consequence, the main conclusion of our work is that guaranteeing the inherent linearizability for these particular executions must incur a high cost in efficiency for a certain class of concurrent, low-contention implementations of (monotone) RMW that are based on switching networks; these are concurrent, low-contention data structures that were recently introduced [7] as a generalization of *counting networks* [2]. Roughly speaking, a *switching network* is a directed, acyclic graph made up of *switches* and *output registers*; whenever a process issues a RMW operation, it shepherds a *token* through the network, which traverses a path of switches till it is eventually returned a value (at an output register). Thus, concurrent processes are spatially dispersed in a switching network, which reduces their simultaneous crossings in front of the same memory location; this offers potential for low contention. The *size* of a switching network is the total number of switches in it; its *latency* is the maximum number of switches traversed by a token shepherding a RMW operation through the network. The *concurrency* of a switching network is the maximum number of concurrent *processes* that may shepherd a RMW operation through the network.

In order to model the low-contention property for switching networks, we introduce *register bottleneck* and *layer bottleneck*; roughly speaking, both register bottleneck and layer bottleneck measure the *minimum* number of network elements (either switches or output registers) that are accessed by processes in any infinite execution. (Layer bottleneck assumes partitioning the switches of the network into *layers* in the natural way.) Intuitively, if this minimum number is small, some network element will become a *bottleneck* (or a “*hot-spot*” in the pool of memory locations) in some infinite execution and the network incurs high contention; hence, a switching network is *low-contention* if register bottleneck and layer bottleneck are sufficiently large.

Contribution. Our chief combinatorial instrument is a *Monotone Linearizability Lemma* (Proposition 5.1), which establishes inherent ordering constraints of linearizability for a certain class of executions of *any* distributed system that implements a monotone RMW operation. Interestingly, in these executions, the arguments of the operations performed by the concurrent processes enjoy together the group-theoretic property of n -wise independence over the associated monotone group.

The end results of our study are negative; they are shown through a modular use of the Monotone Linearizability Lemma. The corresponding general methodology we propose for showing impossibility results for any given class of distributed implementations of monotone RMW operations is to devise suitable counterexample executions which, on one hand, are inherently linearizable (courtesy of the Monotone Linearizability Lemma), while, on the other hand, they are suitably constructed so as to compromise linearizability and thus contradict the Monotone Linearizability Lemma. Applying this general methodology to implementations based on switching networks yields the *first* lower bounds on size and latency for a low-contention switching network that implements a monotone RMW operation. We obtain the following results for any switching network other than the trivial single-switch one:

- If the switching network is made up of *switches* with *finite* state and it is low-contention, then it must contain an *infinite* number of switches, even if concurrency is restricted to remain *bounded* (Theorem 6.1).
- If the switching network is made up of switches with *infinite* state and it is low-contention, then it must still contain an *infinite* number of switches if concurrency is now allowed to grow unbounded (Theorem 6.2).

We note that our two lower bounds on the size of any switching network that implements a monotone RMW operation represent a trade-off between the strength of the switches (finite or infinite state) and the concurrency of the network (bounded or unbounded). Thus, neither of them is implied by the other. Our final result deals with latency. We obtain:

- Any switching network (whether made up of switches of finite or infinite state) that implements a monotone RMW operation induces executions with latency at least $\left\lceil \frac{n-1}{c-1} \right\rceil$, where n is the number of concurrent processes participating in the execution, and c , the network's *capacity*, is the maximum number of processes that simultaneously access a switch in any execution of the network.

Our impossibility results for switching networks indicate that inherent linearizability, necessitated by our Monotone Linearizability Lemma, is the crucial bottleneck that rules out efficiency (with respect to both size and latency) for any low-contention switching network that implements a monotone RMW operation. In fact, we believe that inherent linearizability is indeed the crucial efficiency bottleneck for *any* such class of distributed, low-contention implementations, but this remains to be seen. Finally, we remark that linearizability has so far been studied as a *required* property for a distributed system that best guarantees acceptable concurrent behavior. To the best of our knowledge, our work is the *first* to provide, through the Monotone Linearizability Lemma, a (non-trivial) instance of a distributed system where linearizability is an *inherent* property.

Related Work, Comparison and Significance. The notion of linearizability has been introduced by Herlihy and Wing [12]. Switching networks (and, in particular, *adding networks*) were recently studied in [7], as an extension to counting networks [2] that accommodates the general **Fetch&Add** operation (as opposed to the **Fetch&Increment** and **Fetch&Decrement** operations that were supported before by counting networks [1, 2, 18]); for more on counting networks, see, e.g., [4, 5, 10, 16, 17].

Theorems 6.1 and 6.2 settle the negative a far generalization of a specific open question articulated in [7, Section 5] about the existence of switching networks with a *finite* number of switches that implement the (monotone) **Fetch&Add** operation. (Two solutions, called *adding networks*, with an *infinite* number of switches were presented in [7, Section 4].) Indeed, the more general problem of devising *finite* network-based data structures, as suitable extensions to counting networks, to support synchronization operations other than **Fetch&Increment** (which was originally supported by counting networks) was already stated in the seminal work of Aspnes *et al.* [2] that introduced counting networks; however, it has remained essentially open: progress on this problem has been so far limited to discovering that counting networks themselves

can also support Fetch&Decrement (concurrently with Fetch&Increment)[1, 18]. The impossibility results established in Theorems 6.1 and 6.2 provide a mathematical explanation for the apparent lack of progress on this problem; thus, they are significant since they explain the observed inability of researchers in the last decade or so (since the original conference publication of counting networks [2]) to operationally extend counting networks, while still retaining them finite and low-contention, in order to perform tasks more complex than just incrementing a counter by one but yet as simple as adding an arbitrary value to a counter.

The structure of the proofs of Theorems 6.1 and 6.2 is inspired by that of the proof of a result of Herlihy *et al.* [10, Theorem 5.1], showing that any (non-blocking) counting network [2] (other than the trivial single-balancer one) must have an *infinite* number of balancers if all of its executions are to be linearizable. The requirement that *all* executions be linearizable allows the proof of [10, Theorem 5.1] to pick the execution of choice and force it to violate linearizability. However, a switching network for a monotone RMW operation need not guarantee linearizability in all executions; thus, the role of the Monotone Linearizability Lemma is to contribute to the proofs of Theorems 6.1 and 6.2 executions that are necessarily linearizable. Note also that although a counting network is a special case of a switching network, the lower bound on size established in [10, Theorem 5.1] for a linearizable counting network does not immediately apply to switching networks that implement a monotone RMW operation, since the proof of [10, Theorem 5.1] relies on the behavior of counting networks; instead, the proofs of Theorems 6.1 and 6.2 require far more delicate arguments that are specific to the behavior of switching networks.

Theorem 6.3 is reminiscent of a recent result [7, Theorem 1] that establishes a lower bound of $\left\lceil \frac{n-1}{c-1} \right\rceil$ on latency for adding networks, where there are only two possible arguments for addition, namely a and b such that $|a| > |b| > 0$; more specifically, it is shown that each token of weight b traverses at least $\left\lceil \frac{n-1}{c-1} \right\rceil$ switches, while, if also $|b| > 1$, each token of weight a traverses at least $\left\lceil \frac{n-1}{c-1} \right\rceil$ switches. Theorem 6.3 significantly extends and improves [7, Theorem 1] in the following ways: First, Theorem 6.3 applies to switching networks that implement *any* monotone RMW operation, while [7, Theorem 1] is specific to adding networks and the Fetch&Add operation, and second, despite the enhanced generality of Theorem 6.3, its proof is far simpler and more natural and succinct than that of [7, Theorem 1].

2 Monotone Groups

Basic Definitions. We start by reviewing some very basic definitions from Group Theory. (See [9] for a general background in Group Theory.) A (binary) *operator* (also called *composition law*) on a set \mathbb{F} is a mapping $\oplus : \mathbb{F} \times \mathbb{F} \rightarrow \mathbb{F}$. A *group* $\langle \mathbb{F}, \oplus \rangle$ is a set \mathbb{F} together with an operator \oplus such that: (1) *Closure Property*: for all pairs of elements $a, b \in \mathbb{F}$, $a \oplus b \in \mathbb{F}$, (2) *Associativity*: for all triples of elements $a, b, c \in \mathbb{F}$, $(a \oplus b) \oplus c = a \oplus (b \oplus c)$, (3) *Identity Element*: there is an element $e \in \mathbb{F}$, called the *identity element* of \mathbb{F} , such that for each element $a \in \mathbb{F}$, $a \oplus e = e \oplus a = a$, and (4) *Inverse Element*: for each element $a \in \mathbb{F}$, there is an element $a^{-1} \in \mathbb{F}$, called the *inverse* of a , such that $a \oplus a^{-1} = a^{-1} \oplus a = e$. An *Abelian group* is a group $\langle \mathbb{F}, \oplus \rangle$ which satisfies in addition the following property: (5) *Commutativity*: for all pairs of elements $a, b \in \mathbb{F}$, $a \oplus b = b \oplus a$.

Composite Operators. We proceed to define two composite operators by applying the operator \oplus a number of times. For any integer k , define the unary operator $\bigoplus_k : \mathbb{F} \rightarrow \mathbb{F}$ as follows: $\bigoplus_k a = a \oplus a \oplus \dots \oplus a$ k times if $k > 0$, e if $k = 0$, and $a^{-1} \oplus a^{-1} \oplus \dots \oplus a^{-1}$ $-k$ times if $k < 0$. Call \bigoplus_k the *power operator*. For any integer $n \geq 2$, the operator \biguplus_n is n -ary; it takes as input a sequence of elements a_1, a_2, \dots, a_n , and it yields the result $\biguplus_n(a_1, a_2, \dots, a_n) = a_1 \oplus a_2 \oplus \dots \oplus a_n$, denoted as $\biguplus_{i=1}^n a_i$. (By associativity, the result is well defined.) Call \biguplus the *summation operator*.

Monotone Groups. Assume now that the set \mathbb{F} is totally ordered; thus, a *total order* \preceq is defined on \mathbb{F} . For any pair of elements $a, b \in \mathbb{F}$, write $a \prec b$ if $a \preceq b$ and $a \neq b$. A *monotone subdomain* of \mathbb{F} is a subset $\mathbb{M} \subseteq \mathbb{F}$ that satisfies the following three properties: (1) *Closure*: for any two elements $a, b \in \mathbb{M}$, $a \oplus b \in \mathbb{M}$, (2) *Identity Lower Bound*: for any element $a \in \mathbb{M}$, $e \prec a$, and (3) *Monotonicity under Composition*: for any pair of elements $a, b \in \mathbb{M}$, both $a \prec a \oplus b$ and $b \prec a \oplus b$. Notice that $e \notin \mathbb{M}$. Notice also that \mathbb{M} is

necessarily infinite. A *monotone group* is a quadruple $\langle \mathbb{F}, \mathbb{M}, \oplus, \preceq \rangle$, where $\langle \mathbb{F}, \oplus \rangle$ is an Abelian group, \preceq is a total order on \mathbb{F} , and $\mathbb{M} \subseteq \mathbb{F}$ is a monotone subdomain of \mathbb{F} .

We proceed with some examples of monotone groups that will be used in our later analysis. Throughout, denote \mathbb{Z} , \mathbb{N} and \mathbb{Q} the sets of integers, natural numbers (including zero), and rational numbers, respectively. We will use $+$ and \cdot to denote the common (binary) operators of addition and multiplication, respectively, on these sets. Denote \leq the *less-than-or-equal* relation (total order) on these sets. The quadruple $\langle \mathbb{Z}, \mathbb{N} \setminus \{0\}, +, \leq \rangle$ is a monotone group (*integers with addition*). From the definition of the power operator \bigoplus_k , for any integer k , we have that for any integer $a \in \mathbb{Z}$, $\bigoplus_k a = k \cdot a$. From the definition of the summation operator $\biguplus_{k_1}^{k_2}$, for any pair of integers k_1 and k_2 , we have that for any sequence of $k_2 - k_1 + 1$ integers $a_{k_1}, a_{k_1+1}, \dots, a_{k_2} \in \mathbb{Z}$, $\biguplus_{i=k_1}^{k_2} a_i = \sum_{i=k_1}^{k_2} a_i$. The quadruple $\langle \mathbb{Q}, \mathbb{N} \setminus \{0, 1\}, \cdot, \leq \rangle$ is also a monotone group (*integers with multiplication*). From the definition of the power operator \bigoplus_k , for any integer k , we have that for any rational number $a \in \mathbb{Q}$, $\bigoplus_k a = a^k$. From the definition of the summation operator \biguplus , for any set of n integers k_1, k_2, \dots, k_n , we have that for any set of n rational numbers $a_{k_1}, a_{k_1+1}, \dots, a_{k_2} \in \mathbb{Q}$, $\biguplus_{i=k_1}^{k_2} a_i = \prod_{i=k_1}^{k_2} a_i$.

Independence. Fix any integer $n \geq 2$, and consider any n distinct elements $a_1, a_2, \dots, a_n \in \mathbb{F}$ with $a_1, a_2, \dots, a_n \neq e$. Say that a_1, a_2, \dots, a_n are *n-wise independent* in $\langle \mathbb{F}, \oplus \rangle$ if for any sequence of n integers k_1, k_2, \dots, k_n , where $-1 \leq k_i \leq 2$ for $1 \leq i \leq n$, that are *not all simultaneously zero*, $\biguplus_{i=1}^n \bigoplus_{k_i} a_i \neq e$. Say that the monotone group $\langle \mathbb{F}, \mathbb{M}, \oplus, \preceq \rangle$ is *n-wise independent* if there are n distinct elements $a_1, a_2, \dots, a_n \in \mathbb{M}$, which are *n-wise independent* in $\langle \mathbb{M}, \oplus \rangle$.

By the definition of *n-wise independence*, n integers $a_1, a_2, \dots, a_n \in \mathbb{N} \setminus \{0\}$, where $n \geq 2$, are *n-wise independent* in $\langle \mathbb{N} \setminus \{0\}, + \rangle$ if for any sequence of n integers $k_1, k_2, \dots, k_n \in \{-1, 0, 1, 2\}$, that are not all simultaneously zero, $\sum_{i=1}^n k_i a_i \neq 0$. We are able to prove that for any integer $n \geq 2$, the monotone group $\langle \mathbb{Z}, \mathbb{N} \setminus \{0\}, +, \leq \rangle$ is *n-wise independent*. From the definition of *n-wise independence*, n integers $a_1, a_2, \dots, a_n \in \mathbb{N} \setminus \{0, 1\}$ are *n-wise independent* in $\langle \mathbb{N} \setminus \{0, 1\}, \cdot \rangle$ if for any sequence of n integers $k_1, k_2, \dots, k_n \in \{-1, 0, 1, 2\}$, that are not all simultaneously zero, $\prod_{i=1}^n a_i^{k_i} \neq 1$. Consider any n distinct prime numbers a_1, a_2, \dots, a_n . Then, $\prod_{i=1}^n a_i^{k_i}$ is a rational number whose numerator and denominator have no common factors; so $\prod_{i=1}^n a_i^{k_i} \neq 1$, and the n integers a_1, a_2, \dots, a_n are *n-wise independent* in $\mathbb{N} \setminus \{0, 1\}$. This implies that the monotone group $\langle \mathbb{Q}, \mathbb{N} \setminus \{0, 1\}, \cdot, \leq \rangle$ is *n-wise independent*.

Independence of Monotone Groups. We are able to show that *every* monotone group is *n-wise independent*. The proof uses a reduction to the (already proven) *n-wise independence* of the monotone group $\langle \mathbb{Q}, \mathbb{N} \setminus \{0\}, +, \leq \rangle$. (Thus, this establishes some kind of *completeness* of the monotone group $\langle \mathbb{Q}, \mathbb{N} \setminus \{0\}, +, \leq \rangle$ for the class of all *n-wise independent* monotone groups.)

3 System Model

Systems that Implement Monotone Groups. Our model of a distributed system is patterned after the one in [12, Section 2], adjusted to incorporate the issue of implementing a monotone group $\langle \mathbb{F}, \mathbb{M}, \oplus, \preceq \rangle$. We consider a distributed system \mathbf{P} consisting of a collection of sequential threads of control, called *processes*. Processes are sequential, and each process applies a sequence of operations to a distributed data structure, called the *object*, alternately issuing an invocation and then receiving the associated response. Each *invocation* at process p_i has the form $\text{Invoke}_i(a)$ for some value $a \in \mathbb{M}$; each *response* at process p_i has the form $\text{Response}_i(b)$ for some value $b \in \mathbb{M} \cup \{e\}$. Formally, an *execution* of system \mathbf{P} is a (possibly infinite) sequence α of *invocation* and *response* events. We assume that for each invocation at process p_i in execution α , there is a later response in α that matches it and no invocation at p_i that precedes the matching response in α . An *operation* at process p_i in execution α is a matching pair $op_i = [\text{Invoke}_i(a), \text{Response}_i(b)]$ of an invocation and response at p_i ; for such an operation, we will write $a = \text{In}(op_i)$ and $b = \text{Out}(op_i)$, and we will sometimes say that op_i is of *type* a .

An execution α induces a partial order $\xrightarrow{\alpha}$ on the set of operations in α as follows: For any two operations $op_{i_1} = [\text{Invoke}_{i_1}(a_1), \text{Response}_{i_1}(b_1)]$ and $op_{i_2} = [\text{Invoke}_{i_2}(a_2), \text{Response}_{i_2}(b_2)]$ at processes p_{i_1} and p_{i_2} , respectively, say that op_{i_1} *precedes* op_{i_2} in execution α , denoted $op_{i_1} \xrightarrow{\alpha} op_{i_2}$, if the response $\text{Response}_{i_1}(b_1)$

precedes the invocation $\text{Invoke}_{i_2}(a_2)$. In particular, execution α induces, for each process p_i a total order $\xrightarrow{\alpha}_i$ on the set of operations at p_i in α as follows: For any two operations $op_i^{(1)}$ and $op_i^{(2)}$, $op_i^{(1)} \xrightarrow{\alpha}_i op_i^{(2)}$ if and only if $op_i^{(1)} \xrightarrow{\alpha} op_i^{(2)}$. If, in execution α , operation op_{i_1} does not precede operation op_{i_2} , then we write $op_{i_1} \not\xrightarrow{\alpha} op_{i_2}$. If simultaneously $op_{i_1} \not\xrightarrow{\alpha} op_{i_2}$ and $op_{i_2} \not\xrightarrow{\alpha} op_{i_1}$, then we say that op_{i_1} and op_{i_2} are *parallel* in execution α , denoted as $op_{i_1} \parallel_{\alpha} op_{i_2}$.

For any execution α of system \mathbf{P} , a *serialization* $S(\alpha)$ of execution α is a sequence whose elements are the operations of α , and each operation of α appears exactly once in $S(\alpha)$. Thus, a serialization $S(\alpha)$ is a total order $\xrightarrow{S(\alpha)}$ on the set of operations in α . Notice that there may be, in general, many possible serializations of the execution α . Say that a serialization $S(\alpha)$ is *valid for the monotone group* $\langle \mathbb{F}, \mathbb{M}, \oplus, \preceq \rangle$ if the following two conditions hold: (1) *Valid Start*: if $op_i = [\text{Invoke}_i(a), \text{Response}_i(b)]$ is the first operation in $S(\alpha)$, then $b = e$, and (2) *Valid Composition*: for any pair of operations $op_{i_1}^{(1)} = [\text{Invoke}_{i_1}(a_1), \text{Response}_{i_1}(b_1)]$ and $op_{i_2}^{(2)} = [\text{Invoke}_{i_2}(a_2), \text{Response}_{i_2}(b_2)]$ that are consecutive in $S(\alpha)$, $b_2 = b_1 \oplus a_1$. Say that \mathbf{P} *implements the monotone group* $\langle \mathbb{F}, \mathbb{M}, \oplus, \preceq \rangle$ if every execution of α has a valid serialization. We prove the *Unique Serialization Lemma*, asserting that for any execution α of \mathbf{P} implementing a monotone group, there is a *unique* valid serialization $S(\alpha)$. Sometimes, we will write $\text{In}_{\alpha}(op)$ and $\text{Out}_{\alpha}(op)$ in order to emphasize reference to execution α .

Linearizable Executions. We consider a system \mathbf{P} that implements a monotone group $\langle \mathbb{F}, \mathbb{M}, \oplus, \preceq \rangle$. Say that execution α is *linearizable* [12] if the (unique) valid serialization $S(\alpha)$ extends $\xrightarrow{\alpha}$; that is, for any pair of operations $op^{(1)}$ and $op^{(2)}$ such that $op^{(1)} \xrightarrow{\alpha} op^{(2)}$, $op^{(1)} \xrightarrow{S(\alpha)} op^{(2)}$. Since \mathbf{P} implements the monotone group $\langle \mathbb{F}, \mathbb{M}, \oplus, \preceq \rangle$, for any two operations $op^{(1)}$ and $op^{(2)}$ such that $op^{(1)} \xrightarrow{S(\alpha)} op^{(2)}$, $\text{Out}(op^{(1)}) \prec \text{Out}(op^{(2)})$. Thus, it follows that for any pair of operations $op^{(1)}$ and $op^{(2)}$ such that $op^{(1)} \xrightarrow{\alpha} op^{(2)}$, $\text{Out}(op^{(1)}) \prec \text{Out}(op^{(2)})$. Say that operation $op^{(1)}$ in execution α is *non-linearizable in execution* α if there is another operation $op^{(2)}$ in execution α such that $op^{(2)} \xrightarrow{\alpha} op^{(1)}$ while $\text{Out}(op^{(1)}) \prec \text{Out}(op^{(2)})$. Say that operation op in execution α is *linearizable in execution* α if it is not non-linearizable in execution α . Clearly, execution α is linearizable if every operation in execution α is linearizable in it.

4 Switching Networks

Basic Definitions. A *switching network* [7] is a directed acyclic graph in which the nodes are called *switches* and the edges are called *wires*. An (f_{in}, f_{out}) -*switch* is a routing element with f_{in} *input wires*, f_{out} *output wires*, and an *internal state*. A (w_{in}, w_{out}) -*switching network* has w_{in} *input wires* and w_{out} *output wires*, and it is formed by connecting together switches; thus, we connect output wires of switches to input wires of other switches. Some switches have input wires (resp., output wires) not connected to other switches in the network, and these wires are called the *input wires* (resp., *output wires*) of the network. The *size* of a switching network is the number of its switches. A *path* in a switching network is a sequence of switches, each connected to the next. The *depth* $d(b)$ of a switch b in a switching network is defined to be 0 if one of its input wires is an input wire of the network, and $\max_j d(b_j) + 1$, where the maximum is taken over all switches b_j that are connected to switch b . The *depth* d of the network is defined as the maximum depth of any switch. The switching network can naturally be divided into d *layers*, so that layer ℓ contains all switches of depth ℓ , where $0 \leq \ell \leq d$. A simple (4, 4)-switching network with depth 3, made from (2, 2)-switches, is depicted in Figure 1, where switches are drawn with vertical lines and wires with horizontal lines; for each switch, inputs wires appear on its left and output wires on its right, and similarly for the switching network.

Tokens. Processes access the switching network by issuing *tokens*. In contrast to counting networks [2], each token has a *state* (a set of variables) which can change as the token traverses the network. In particular, a token enters the switching network from one of the network's input wires; then, the token is forwarded to the switch to which the wire belongs, the switch then routes the token to one of its output wires from which the token enters the next switch in the network, and so on. The token continues traversing the network in the same fashion until it reaches an output wire of the network. Then, the token exits the network and returns to the process that issued it. When a token traverses a switch, the states of the token and the

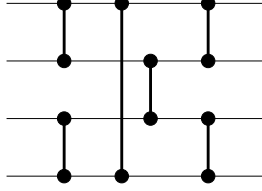


Figure 1: A simple switching network.

switch change atomically before the token is routed to an output wire of the switch. Note that the token and the switch have different transition functions for their states. A switching network may be accessed by many tokens simultaneously which traverse the network asynchronously; however, each process has at most one token traversing the network each time. The *latency* of the switching network is the maximum number of switches traversed by any token (thus, it does not exceed the depth of the network). The *concurrency* of a switching network is the maximum number of processes (hence, tokens) allowed to access the network simultaneously.

Configurations. A *network configuration* of a switching network is the concatenation of the current states of the network's switches. A *total configuration* of a switching network is the concatenation of the current states of the networks' switches and the states of all tokens that are currently traversing the network. Say that a switching network is in a *quiescent* total configuration if there are no tokens traversing the network (that is, all tokens that have entered the network have exited it). Denote x_i the total number of tokens that have ever entered from input wire i of the network, where $1 \leq i \leq w_{in}$, and denote y_j the total number of tokens that have left from output wire j of the network, where $1 \leq j \leq w_{out}$. The network must satisfy the following two properties: (1) *Safety property*: in any total configuration, it must be $\sum_{i=1}^{w_{in}} x_i \geq \sum_{j=1}^{w_{out}} y_j$; thus, no new tokens are created in the network, and (2) *Liveness Property*: given any finite number of input tokens that traverse the network, the network will eventually reach a quiescent total configuration. In any quiescent configuration it must be that $\sum_{i=1}^{w_{in}} x_i = \sum_{j=1}^{w_{out}} y_j$. The safety and liveness properties must also be satisfied by every individual switch in the network.

Executions. We model executions of switching networks in the style of Herlihy *et al.* [10]. For any switch b and token t , we denote by $e = \langle t, b \rangle$ the *transition* in which the token passes (in one atomic step) from an input wire to an output wire of switch b . An *execution* of a switching network is a finite or infinite sequence $s_0, e_1, s_1, e_2, \dots$ of alternating total configurations and switch transitions such that for each triple $\langle s_i, e_{i+1}, s_{i+1} \rangle$, the switch transition e_{i+1} carries the total configuration s_i to total configuration s_{i+1} . A finite execution is *complete* if it results to a quiescent total configuration. An execution α is *sequential* if for any two transitions $e_i = \langle t_i, b_i \rangle$ and $e_j = \langle t_j, b_j \rangle$, where t_i and t_j correspond to the same token, all transitions (if any) between them also involve that token. In other words, tokens traverse the network one completely after the other in a sequential execution. In an execution of a switching network, we say that concurrency is *bounded* if the number of concurrent processes accessing the network in the execution is bounded. In an (infinite) execution, we say that concurrency is *unbounded* if the number of concurrent processes accessing the network in the execution is unbounded (in which case it is either finite or infinite).

Implementations. A switching network \mathcal{N} can be used to implement a monotone group $\langle \mathbb{T}, \mathbb{M}, \oplus, \preceq \rangle$. Each token t issued by process p_i corresponds to an operation $op_i = [\text{Invoke}_i(a), \text{Response}_i(v)]$ invoked by process p_i , where $a \in \mathbb{M}$ and $v \in \mathbb{M} \cup \{e\}$. We say that a is the *input value* of the token t , and v is the *output value* of the token t . The input value of the token is part of the token's initial state. In any execution α , the invocation of operation op corresponds to the first transition $e_i = \langle t_i, b_i \rangle$ where $t_i = t$ and b_i is an input switch of the network (this transition occurs when the token enters the network); the response of op corresponds to the latest transition $e_j = \langle t_j, b_j \rangle$ in execution α such that $t_j = t$ (this transition occurs when the token exits the network). When token t exits the network, it carries encapsulated in its state the output value v that operation op_i responds with. Use execution α to define its subsequence α' that

contains only transitions that correspond to invocations and responses of the operations corresponding to tokens. The sequence α' induces an execution of a distributed system in the natural way. Denote \mathbf{P} the distributed system that is determined by all such induced executions (one for each execution of the switching network \mathcal{N}). Say now that *switching network \mathcal{N} implements the monotone group $\langle \mathbb{I}, \mathbb{M}, \oplus, \preceq \rangle$* if the system \mathbf{P} implements the monotone group $\langle \mathbb{I}, \mathbb{M}, \oplus, \preceq \rangle$.

Finite and Infinite Switches. We examine two kinds of switching networks, corresponding to switches with finite or infinite state.

- *Switching networks with finite switches:* Each switch of the network has a *finite* number of states. For this kind of network, we include an additional component on the output wires of the switching network: the *output registers*. There is an output register associated with each output wire of the switching network. Unlike switches, each output register has an infinite number of states. The output value for a token's operation is computed on the output register residing on the network's output wire from which the token exits. At the exit, the following happen atomically: the token computes its output value according to the register's current state and the state of the register changes according to its previous state and the state of the token (which includes its input value). Notice that the input value of a token does not affect its output value, but only the output value of the next token that will access the same output register.

We remark that this kind of switching networks corresponds more to traditional counting networks [2], where a token fetching the counter's value and incrementing the counter by one obtains the value from the register attached to the output wire it will arrive at. We also remark that output registers are *necessary* for this kind of switching networks, since they provide an infinite number of different output values to tokens, while finite switches, used only for routing, are unable to do so.

- *Switching networks with infinite switches:* Each switch has an infinite number of states. For this kind of networks, there are no attached output registers and the output value of a token is determined according to the state of the token when it exits the network.

Contention Measures. In a switching network, contention represents the extent to which concurrent processes access the same switch or output register simultaneously. We use the following complexity-theoretic measures to model contention in switching networks, the last of which was originally introduced by Dwork *et al.* [6] for the case of counting networks.

- The *register bottleneck* of a switching network \mathcal{N} is the *minimum* number of output registers, the minimum being taken over all infinite executions, accessed by tokens in an infinite suffix of an infinite execution of \mathcal{N} . (This definition applies only to switching networks with finite switches.) Intuitively, a switching network is low-contention if its register bottleneck is large; a register bottleneck of 1 is the *worst*, since then many tokens (as many as processes) may eventually accumulate in front of the same output register, which becomes a "hot-spot".
- Similarly, we define the *layer bottleneck* of a switching network \mathcal{N} to be the *minimum* number of switches in the same layer, the minimum being taken over all layers and infinite executions, accessed by tokens in an infinite suffix of an infinite execution of \mathcal{N} . (This definition will be useful for switching networks with infinite switches.) Intuitively, a switching network is low-contention if its layer bottleneck is large; a layer bottleneck of 1 is the *worst*, since then many tokens (as many as processes) may eventually accumulate in front of the same switch, which becomes a "hot-spot".
- The *capacity c* of a switching network \mathcal{N} is the maximum number of processes that simultaneously access a particular switch in any execution of \mathcal{N} .

5 The Monotone Linearizability Lemma

In this section, we state and prove the Monotone Linearizability Lemma, which establishes ordering constraints of linearizability on a system \mathbf{P} that implements a monotone group $\langle \mathbb{I}, \mathbb{M}, \oplus, \preceq \rangle$. Since the monotone group $\langle \mathbb{I}, \mathbb{M}, \oplus, \preceq \rangle$ is n -wise independent, there exist n distinct elements $a_1, a_2, \dots, a_n \in \mathbb{M}$, with

$a_1, a_2, \dots, a_n \neq e$, which are n -wise independent in $\langle \mathbb{M}, \oplus \rangle$. The proof of the Monotone Linearizability Lemma amounts to establishing a contradiction to n -wise independence for a hypothetical *non-linearizable* execution, in which the arguments of the RMW operations issued by the processes are a_1, a_2, \dots, a_n . We show:

Proposition 5.1 (Monotone Linearizability Lemma) *Consider any execution α of system \mathbf{P} in which each process p_i , $1 \leq i \leq n$, issues only operations of type a_i . Then, α is linearizable.*

Sketch of proof: Assume, by way of contradiction, that α is not linearizable. So, there is at least one operation that is non-linearizable in α . Consider the non-linearizable operation op_k at process p_k with *earliest* response in execution α (among all non-linearizable operations in α). Since op_k is non-linearizable, there exists an operation op_l at some process l such that $op_l \xrightarrow{\alpha} op_k$ while $\text{Out}(op_k) \prec \text{Out}(op_l)$; fix op_l to be the one with *latest* response in execution α . Consider the finite prefix β_1 of execution α that ends with the response for operation op_k . Consider now a finite execution α_1 , which is an extension of β_1 that includes no additional invocations by processes; thus, α_1 contains, in addition to those in β_1 , only responses to invocations that are pending in β_1 . Clearly, all operations whose responses precede or coincide with that of op_k have identical outputs in α and α_1 . This implies, in particular, that the non-linearizable operations in α_1 are op_k and possibly operations with responses following that of op_k (but at most one such operation per process). Take now the (unique) valid serialization $S(\alpha_1)$ of α_1 . Clearly, op_k precedes op_l in $S(\alpha_1)$. For each process p_i , where $1 \leq i \leq n$, denote m_i the number of operations at p_i that precede op_l in the serialization $S(\alpha_1)$. Partition these m_i operations into two classes: (a) Operations whose response in α_1 precedes or coincides with that of op_l ; (thus, op_l falls into this class); assume there are $m_{i,a}$ of them. (b) Operations whose response in α_1 follow that of op_l ; assume there are $m_{i,b}$ of them; thus, $m_i = m_{i,a} + m_{i,b}$. Since each operation at process p_i is of type a_i , and since $S(\alpha_1)$ is a valid serialization of α_1 , we use the associativity and commutativity of operation \oplus to obtain that $\text{Out}(op_l) = \left(\biguplus_{i=1}^n \bigoplus_{m_{i,a}} a_i \right) \oplus \left(\biguplus_{i=1}^n \bigoplus_{m_{i,b}} a_i \right)$. We then prove (by case analysis) that for each process p_i , $0 \leq m_{i,b} \leq 2$.

Consider now the finite prefix β_2 of execution α that ends with the response for operation op_l . Consider a finite execution α_2 , which is an extension of β_2 that includes no additional invocations by processes; thus, α_2 contains, in addition to those in β_2 , only responses to invocations that are pending in β_2 . Clearly, all operations whose responses precede or coincide with that of op_l have identical outputs in α and α_2 . Note also that, by construction, there are no non-linearizable operations in α_2 . Consider the (unique) valid serialization $S(\alpha_2)$ of α_2 . For each process p_i , where $1 \leq i \leq n$, denote m'_i the number of operations at process p_i that precede op_l in the serialization $S(\alpha_2)$. Partition these m'_i operations into two classes: (a) Operations whose response in α_2 precedes or coincides with that of op_l ; (thus, op_l falls into this class); assume there are $m'_{i,a}$ of them. (b) Operations whose response in α_2 follow that of op_l ; assume there are $m'_{i,b}$ of them; thus, $m'_i = m'_{i,a} + m'_{i,b}$. Since each operation at process p_i is of type a_i , and since $S(\alpha_1)$ is a valid serialization of α_1 , we use the associativity and commutativity of operation \oplus to obtain that $\text{Out}(op_l) = \left(\biguplus_{i=1}^n \bigoplus_{m'_{i,a}} a_i \right) \oplus \left(\biguplus_{i=1}^n \bigoplus_{m'_{i,b}} a_i \right)$.

Clearly, all operations whose responses precede or coincide with that of op_l have the same outputs in α and α_1 , and in α and α_2 , respectively. It follows that all such operations have the same outputs in α_1 and α_2 . Consider any such operation op . If op precedes op_l in serialization $S(\alpha_1)$ then it must be that $\text{Out}(op) \prec \text{Out}(op_l)$. Subsequently, it must be that op precedes op_l in serialization $S(\alpha_2)$ too, since otherwise the valid composition condition of the valid serialization for a monotone group would be violated. For the same reason, if op does not precede op_l in serialization $S(\alpha_1)$ then it must be that op does not precede op_l in serialization $S(\alpha_2)$ either. This implies that for all processes p_i , $1 \leq i \leq n$, $m_{i,a} = m'_{i,a}$. We use the fact that operation op_l has the same output in executions α_1 and α_2 to conclude (using the associativity and commutativity of \oplus) that $\biguplus_{i=1}^n \bigoplus_{m_{i,b} - m'_{i,b}} a_i = e$. We next prove that for all processes p_i , where $1 \leq i \leq n$, $0 \leq m'_{i,b} \leq 1$, from which it follows that for all processes p_i , $-1 \leq m_{i,b} - m'_{i,b} \leq 2$. We next prove that the difference $m_{k,b} - m'_{k,b}$ never vanishes. We argue that $m_{k,b} = 1$, and we prove, by case analysis, that $m'_{k,b} = 0$. It therefore follows, from the equation $\biguplus_{i=1}^n \bigoplus_{m_{i,b} - m'_{i,b}} a_i = e$ that a_1, a_2, \dots, a_n are not n -wise independent. A contradiction. \blacksquare

6 Impossibility Results and Lower Bounds

Lower Bounds on Size. We first consider switching networks with finite switches. We show:

Theorem 6.1 (Impossibility Result for Switching Networks with Finite Switches) *There is no non-trivial switching network with finite switches that has finite size, incurs register bottleneck at least 2 and implements a monotone group $\langle \mathbb{F}, \mathbb{M}, \oplus, \preceq \rangle$, when the concurrency is bounded.*

Sketch of proof: Assume, by way of contradiction that there is such a (non-trivial) switching network \mathcal{N} with finite size. Since the number of switches in the network \mathcal{N} is finite, and each switch has a finite number of states, the switching network has a finite number of network configurations, which we denote by s . The number of output registers is also finite, which we denote by m . Consider a sequential execution α of network \mathcal{N} with $s + 1$ tokens. By the pigeon-hole principle, some network configuration of \mathcal{N} is repeated in α ; so take any pair of identical network configurations, and call *phase* the execution segment of α that appears between them. Since the register bottleneck of \mathcal{N} is at least 2, it must be that in any phase ϕ , there exist at least two distinct tokens that access two different output registers (since, otherwise, we would be able to construct an infinite execution with register bottleneck 1 by repeating ϕ infinitely many times, so that only one output register would be accessed in the resulting infinite suffix).

Consider now a sequential execution α_1 involving $3m(s + 1)$ tokens, whose input values are taken to be any $3m(s + 1)$ -wise independent elements in $\langle \mathbb{M}, \oplus \rangle$. Execution α_1 must contain at least $3m$ disjoint phases; since the number of output registers is m , and since at least two output registers are accessed in each phase, it follows that there exists some output register r_1 that is accessed in at least three different phases ϕ_1 , ϕ_2 and ϕ_3 (in this order). In ϕ_1 , register r_1 is accessed by some token t_i . In ϕ_2 , besides r_1 , there must exist another register r_2 which is accessed by some token t_j (since in each phase at least two distinct registers are accessed). In ϕ_3 , register r_1 is accessed by a token t_k . Proposition 5.1 implies that execution α_1 is linearizable, so that $\text{Out}_{\alpha_1}(t_i) \prec \text{Out}_{\alpha_1}(t_j) \prec \text{Out}_{\alpha_1}(t_k)$.

We construct now an execution α_2 which involves the same tokens (with same input values) as α_1 . Execution α_2 is identical with α_1 up to the point where t_i is about to access register r_1 , when the following occur:

Token t_i does not take its transition to r_1 , and we say that t_i is *halted* on r_1 (or that we halt it on r_1). Whenever any subsequent token different than t_k attempts to access r_1 , we halt it on r_1 , otherwise we let it exit the network. When it is t_k 's turn, we let it take its transition step on r_1 . We then let t_i take its transition on r_1 , and then the rest of the halted tokens take their transitions on r_1 . (The order of tokens is the same as in α_1 .)

Notice that in execution α_2 , each token accesses the same output register as in α_1 . Thus, it follows that token t_j does not access register r_1 in execution α_2 (but it accesses register r_2), and thus it returns the same output value as in α_1 ; hence, $\text{Out}_{\alpha_1}(t_j) = \text{Out}_{\alpha_2}(t_j)$. Since in executions α_1 and α_2 , tokens t_i and t_k “see” the same state of register r_1 (that is, token t_k bypasses the halted token t_i and takes on the value it takes in α_1), $\text{Out}_{\alpha_1}(t_i) = \text{Out}_{\alpha_2}(t_k)$. It follows that $\text{Out}_{\alpha_2}(t_k) \prec \text{Out}_{\alpha_2}(t_j)$. However, from Proposition 5.1, execution α_2 is linearizable, while by construction of α_2 , token t_j completely precedes token t_k in α_2 ; it follows that $\text{Out}_{\alpha_2}(t_j) \prec \text{Out}_{\alpha_2}(t_k)$. A contradiction. ■

We remark that the concurrency assumed in the proof of Theorem 6.1 is no more than the number of tokens involved in the proof, which is a *bounded* quantity ($3m(s + 1)$) depending only on parameters of the network \mathcal{N} . Thus, the impossibility result in Theorem 6.1 holds even for networks with bounded concurrency. Finally, we argue that the assumption of a *non-trivial* switching network is essential for Theorem 6.1 to hold: since each token can atomically invoke a computation on an output register, we can implement a monotone RMW operation by a *trivial* switching network consisting of a single switch that outputs tokens along one output wire, which has an associated register that maintains the state of the RMW variable to be implemented. The switch sequences the operations (that correspond to the tokens) so that they can be atomically invoked (by the tokens) on the register.

We now turn to switching networks with infinite switches. Clearly, the proof of Theorem 6.1 is not applicable to switching networks with infinite switches, since the number of their possible network configurations is no longer finite. Thus, we need to develop new techniques to handle them. We show:*

*The proof of Theorem 6.2 will consider (without loss of generality) so called *normalized* switching networks, in which any switch b at layer ℓ has its input wires connected to switches of layer $\ell - 1$ (assuming $\ell \geq 2$) and its output wires connected to

Theorem 6.2 (Impossibility Result for Switching Networks with Infinite Switches) *There is no non-trivial switching network with infinite switches that has finite size, incurs layer bottleneck at least 2 and implements a monotone group $\langle \mathbb{F}, \mathbb{M}, \oplus, \preceq \rangle$, assuming concurrency is unbounded.*

Sketch of proof: Assume, by way of contradiction that there is such a switching network \mathcal{N} with finite size. Assume that the depth of \mathcal{N} is d , and that \mathcal{N} is partitioned into layers ℓ_1, \dots, ℓ_d . Consider an execution α of the network \mathcal{N} which involves an infinite number of tokens t_1, t_2, \dots , issued by distinct processes. In execution α , token t_1 is the first token which enters and traverses the network \mathcal{N} . Afterwards, each token t_{i+1} , where $i > 1$, enters the network when one of the following two events occurs to the previous token t_i : token t_i “finishes” its traversal in \mathcal{N} by leaving from one of the network’s output wires, or token t_i “halts” on some switch in \mathcal{N} without taking its transition on that switch (as explained below).

Let b_1^1 and b_2^1 denote two distinct switches at the first layer ℓ_1 (the existence of these two distinct switches follows from the fact that the layer bottleneck is at least 2 and the number of tokens in execution α are infinite). In execution α , tokens with odd index enter the network \mathcal{N} from switch b_1^1 (*class-1* tokens), and tokens with even index enter the network from switch b_2^1 (*class-2* tokens). Note that there is an infinite number of *class-1* and *class-2* tokens. In execution α we let each token traverse its respective switch of the first layer. Since the number of tokens of *class-1* is infinite, there is a switch b_1^2 at layer ℓ_2 such that an infinite number of tokens of *class-1* access b_1^2 . Similarly, there is a switch b_2^2 at layer ℓ_2 such that an infinite number of tokens of *class-2* access b_2^2 . We choose the switches b_1^2 and b_2^2 so that are not the same, namely, $b_1^2 \neq b_2^2$. Note that we can always find such distinct switches at layer ℓ_2 , since otherwise, there would be only one switch at layer ℓ_2 accessed by an infinite number of tokens from both classes, which implies that there is an infinite suffix of α that violates the assumption that the layer bottleneck is at least 2.

In execution α , we let all *class-1* tokens that access switch b_1^2 to take their transitions on this switch, and the rest of the *class-1* tokens to halt on their layer ℓ_2 switch (which is different than b_1^2). We treat similarly the *class-2* tokens on switch b_2^2 . Note that there is an infinite number of *class-1* and *class-2* tokens that exit from the second layer. Execution α continues in the same fashion for each subsequent layer. As a consequence, in execution α an infinite number of *class-1* and *class-2* tokens finish by leaving the network from an output wire. The *class-1* tokens that finish follow the path $b_1^1, b_1^2, \dots, b_1^d$, while the *class-2* tokens that do not finish are halted on some switch of this path. Similarly, the *class-2* path is $b_2^1, b_2^2, \dots, b_2^d$. Since $b_1^i \neq b_2^i$, for any i , $1 \leq i \leq d$, we have that in execution α the path of tokens from *class-1* does not intersect with the path of tokens from *class-2*.

Consider now the shortest prefix α' of execution α in which a token t_{i_1} of *class-1* finishes and a token t_{i_2} of *class-2* finishes. Since α' is a prefix of α it must be that the number of tokens in α' is bounded by some number m . We can choose the input values of the first m tokens of prefix α' so that they are m -wise independent.

Next, we construct a bounded execution β from the prefix α' , such that in β all the *class-1* tokens enter the network \mathcal{N} before the *class-2* tokens enter the network (within each class, the relative order of tokens is the same as in α'). Notice that in prefix α' and in execution β the tokens behave in exactly the same way, since the paths of the tokens from the different classes do not intersect. Proposition 5.1 implies that execution β is linearizable, since the input values of the tokens are m -wise independent. Therefore, since in execution β the token t_{i_2} enters the network after token t_{i_1} finishes, it must be $\text{Out}_\beta(t_{i_1}) \prec \text{Out}_\beta(t_{i_2})$. Similar to β , we can construct an execution γ where now the *class-2* tokens enter the network before the *class-1* tokens. Execution γ must be linearizable, and since token t_{i_1} enters the network after token t_{i_2} finishes, it must be that $\text{Out}_\gamma(t_{i_2}) \prec \text{Out}_\gamma(t_{i_1})$. Notice that all the tokens in executions α' , β and γ behave in exactly the same way, since tokens from *class-1* and *class-2* do not intersect. Therefore, $\text{Out}_\beta(t_{i_1}) = \text{Out}_\gamma(t_{i_1})$ and $\text{Out}_\beta(t_{i_2}) = \text{Out}_\gamma(t_{i_2})$. Subsequently, $\text{Out}_\beta(t_{i_2}) \prec \text{Out}_\beta(t_{i_1})$. A contradiction. \blacksquare

We remark that the proof of Theorem 6.2 requires unbounded (finite or infinite) concurrency. This stems from the assumption that the layer bottleneck is at least 2; thus, in any infinite suffix of an infinite execution, at least two switches in each layer are certain to be accessed by tokens. We do not know, however, how many tokens we will need (and, therefore, how many processes) before a second switch is accessed; hence,

switches of layer $\ell + 1$ (assuming ℓ is less than the depth of the network). Thus, in a normalized switching network, there are no wires connecting switches in non-consecutive layers. Note that any switching network can be easily cast as a normalized one, if we intercept wires that connect non-consecutive layers with *dummy* switches with input and output width 1, which simply forward tokens (without routing them).

we cannot bound the concurrency needed in the execution used in the proof. This was not the case for switching networks with finite switches (even though the assumption on register bottleneck made there is quite similar), where the fact that configurations were repeated (since switches are finite) allowed to get a token visit a register for a second time by shepherding in a finite number of tokens. So, since the proof of Theorem 6.2 requires unbounded concurrency, it does *not* imply Theorem 6.1 which assumes bounded concurrency, and the two results are incomparable and represent a trade-off.

Finally, we remark that the assumption of a *non-trivial* switching network is essential for Theorem 6.2 to hold: A switching network consisting of a single infinite-state switch with n input wires and n output wires (where n is the number of concurrent processes) can implement any RMW variable as follows. The state of the variable is encoded by the state of the switch. To invoke an operation on the variable, a process issues a token with a state encoding the argument of the operation. Such a token, when atomically processed by the switch, will cause the natural changes to its state and to the state of the switch, so that the new state of the switch is the new state of the variable, and the new state of the token is the response of the variable to the operation invoked by the token.

Lower Bound on Time. We start with a definition that we will use in our proof. For any quiescent total configuration s of a switching network \mathcal{N} , we say that token t_i has *preferred path* π if t_i follows the path π and runs in isolation into the network, which is initially in the total configuration s , until token t_i exits the network and responds with an output value v which is its *preferred value*. We show:

Theorem 6.3 *For any switching network \mathcal{N} that implements a monotone group $\langle \mathbb{I}, \mathbb{M}, \oplus, \preceq \rangle$, there is a sequential execution with n tokens such that each token traverses at least $\left\lceil \frac{n-1}{c-1} \right\rceil$ switches.*

Proof: Consider n tokens t_1, t_2, \dots, t_n issued by distinct processes, with respective input values $a_1, a_2, \dots, a_n \in \mathbb{M}$ (trivially, $a_1, a_2, \dots, a_n \neq e$) which are n -wise independent in $\langle \mathbb{M}, \oplus \rangle$. First we show that the preferred paths of any two tokens, starting from the same total quiescent configuration, intersect. Consider the network \mathcal{N} in a quiescent configuration s . Denote v the output value returned by \mathcal{N} to the last token in the (unique) valid serialization of the execution fragment ending with total configuration s , and let a denote that last token's input value. Consider tokens t_i and t_j with input values a_i and a_j , respectively. Assume, by way of contradiction, that the preferred paths of t_i and t_j starting from total configuration s do not intersect. Since the network \mathcal{N} provides a valid implementation of the monotone group $\langle \mathbb{I}, \mathbb{M}, \oplus, \preceq \rangle$, and since the preferred paths do not intersect, the output values of t_i and t_j when they run sequentially into the network \mathcal{N} with t_i first and t_j next, starting from s are equal to $v \oplus a$ (those that would be returned in separate executions where only one of the tokens would be running). However, execution α' , where both tokens are running, is linearizable (by Proposition 5.1). Hence, the token t_i is serialized before token t_j in the (unique) valid serialization of execution α' . Since the network \mathcal{N} provides a valid implementation of the monotone group $\langle \mathbb{I}, \mathbb{M}, \oplus, \preceq \rangle$, the output values of t_i and t_j are $v \oplus a$ and $v \oplus a \oplus a_i$, respectively. Since $a_i \neq e$, $v \oplus a \neq v \oplus a \oplus a_i$, a contradiction. It follows that the preferred paths of t_i and t_j starting from total configuration s do intersect. By the definition of c , no more than $c - 1$ tokens (other than t_i) can access any switch along the preferred path of t_i (starting from total configuration s). Since every other process's preferred path must intersect t_i 's preferred path, it follows that the preferred path of t_i must include at least $\left\lceil \frac{n-1}{c-1} \right\rceil$ switches. Take now any sequential execution α of \mathcal{N} , starting from any arbitrary quiescent total configuration s , in which each token t_i issues only operations with input value a_i . Since the preferred path of any token includes at least $\left\lceil \frac{n-1}{c-1} \right\rceil$ switches, the first token to run will traverse at least $\left\lceil \frac{n-1}{c-1} \right\rceil$ switches of \mathcal{N} , and the network will return to another quiescent total configuration, for which the same argument applies inductively. ■

7 Conclusion and Open Problems

We have studied the possibility, and the corresponding costs, of implementing a monotone RMW operation in a concurrent and low-contention manner. Our end results are lower bounds on size and latency for any non-trivial, low-contention switching network that implements a monotone RMW operation; these are shown by using the Monotone Linearizability Lemma, which may be of independent interest. It would be interesting

to ask whether *timing conditions* may suffice to overcome the limitations we have shown; recall that timing conditions have been exploited in the work of Lynch *et al.* [16] for devising *finite-size* linearizable counting networks, while Herlihy *et al.* [10] establish that no finite-size (non-trivial) *asynchronous* linearizable counting network exists. For future work, we are also interested in establishing further limitations on various kinds of distributed systems (other than switching networks) that implement a monotone RMW operation. A natural candidate to consider is the message-passing system adopted in the work by Wattenhofer and Widmayer [19]; that work showed a lower bound on the message complexity of implementing the Fetch&Increment operation in that system; we feel that similar limitations hold for implementations of *any* monotone RMW operation.

References

- [1] W. Aiello, C. Busch, M. Herlihy, M. Mavronicolas, N. Shavit and D. Touitou, “Supporting Increment and Decrement Operations in Balancing Networks,” *Chicago Journal of Theoretical Computer Science*, 2000-4, December 14, 2000 (electronic). Preliminary version in *Proceedings of the 16th International Symposium on Theoretical Aspects of Computer Science (STACS 1999)*, Vol. 1563, Lecture Notes in Computer Science, Springer-Verlag, pp. 377–386, 1999.
- [2] J. Aspnes, M. Herlihy and N. Shavit, “Counting Networks,” *Journal of the ACM*, Vol. 41, pp. 1020–1048, 1994.
- [3] H. Attiya and J. L. Welch, *Distributed Computing: Fundamentals, Simulations and Advanced Topics*, McGraw-Hill, 1998.
- [4] C. Busch and M. Herlihy, “Sorting and Counting Networks of Small Depth and Arbitrary Width,” *Theory of Computing Systems*, Vol. 35, No. 2, pp. 99–128, 2002.
- [5] C. Busch and M. Mavronicolas, “A Combinatorial Treatment of Balancing Networks,” *Journal of the ACM*, Vol. 43, No. 5, pp. 794–839, September 1996.
- [6] C. Dwork, M. Herlihy and O. Waarts, “Contention in Shared Memory Algorithms,” *Journal of the ACM*, Vol. 44, No. 6, pp. 779–805, November 1997.
- [7] P. Fatourou and M. Herlihy, “Adding Networks,” *Proceedings of the 15th International Symposium on Distributed Computing (DISC 2001)*, J. L. Welch ed., pp. 330–342, Vol. 2180, Lecture Notes in Computer Science, Springer-Verlag, Lisbon, Portugal, October 2001.
- [8] J. Goodman, M. Vernon and P. Woest, “Efficient Synchronization Primitives for Large-Scale, Cache-Coherent Multiprocessors,” *Proceedings of the 3rd International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 64–75, April 1989.
- [9] M. Hall, Jr., *Theory of Groups*, Second Edition, Chelsea Publishing Company, 1979.
- [10] M. Herlihy, N. Shavit and O. Waarts, “Linearizable Counting Networks,” *Distributed Computing*, Vol. 9, pp. 193–203, 1996.
- [11] M. Herlihy, S. Tirthapura and R. Wattenhofer, “Ordered Multicast and Distributed Swap,” *Operating Systems Review*, Vol. 35, No. 1, pp. 85–96, January 2001.
- [12] M. Herlihy and J. Wing, “Linearizability: A Correctness Condition for Concurrent Objects,” *ACM Transactions on Programming Languages and Systems*, Vol. 12, No. 3, pp. 463–492, 1990.
- [13] C. P. Kruskal, L. Rudolph and M. Snir, “Efficient Synchronization on Multiprocessors with Shared Memory,” *Proceedings of the 5th Annual ACM Symposium on Principles of Distributed Computing*, pp. 218–228, August 1986.
- [14] L. Lamport, “How to Make a Multiprocessor Computer that Correctly Executes Multiprocess Programs,” *IEEE Transactions on Computers*, Vol. C-28, No. 9, pp. 690–691, September 1979.
- [15] N. Lynch, *Distributed Algorithms*, Morgan Kaufmann, 1996.
- [16] N. Lynch, N. Shavit, A. Shvartsman and D. Touitou, “Timing Conditions for Linearizability in Uniform Counting Networks,” *Theoretical Computer Science*, Vol. 220, No. 1, pp. 67–91, June 1999.
- [17] M. Mavronicolas, M. Merritt and G. Taubenfeld, “Sequentially Consistent versus Linearizable Counting Networks,” *Proceedings of the 18th Annual ACM Symposium on Principles of Distributed Computing*, pp. 133–142, May 1999.
- [18] N. Shavit and D. Touitou, “Elimination Trees and the Construction of Pools and Stacks,” *Theory of Computing Systems*, Vol. 30, No. 6, pp. 545–570, November/December 1997.
- [19] R. Wattenhofer and P. Widmayer, “An Inherent Bottleneck in Distributed Counting,” *Journal of Parallel and Distributed Computing*, Vol. 49, pp. 135–145, 1998.

Appendix

A Pairwise Independence

Consider any two distinct elements $a_1, a_2 \in \mathbb{F}$ with $a_1, a_2 \neq e$. Say that a_1 and a_2 are *pairwise independent* in $\langle \mathbb{F}, \oplus \rangle$ if for any pair of integers k_1 and k_2 , both $a_1 \neq \bigoplus_{k_1} a_2$ and $a_2 \neq \bigoplus_{k_2} a_1$. Thus, neither a_1 nor a_2 may result by repetitive application of the operator on the other or on the inverse of the other. Say that the monotone group $\langle \mathbb{F}, \mathbb{M}, \oplus, \preceq \rangle$ is *pairwise independent* if there are two distinct elements $a_1, a_2 \in \mathbb{M}$, with $a_1, a_2 \neq e$, which are pairwise independent in $\langle \mathbb{M}, \oplus \rangle$.

By the definition of pairwise independence, two integers $a_1, a_2 \in \mathbb{N} \setminus \{0\}$, with $a_1, a_2 \neq e$, are pairwise independent in $\langle \mathbb{N} \setminus \{0\}, + \rangle$ if for any integer k , both $a_1 \neq ka_2$ and $a_2 \neq ka_1$. It is straightforward to see that any two consecutive integers a and $a + 1$, where $a \geq 2$, are pairwise independent, which immediately implies that the monotone group $\langle \mathbb{Z}, \mathbb{N} \setminus \{0\}, +, \leq \rangle$ is pairwise independent.

From the definition of pairwise independence, two integers $a_1, a_2 \in \mathbb{N} \setminus \{0, 1\}$ are pairwise independent in $\langle \mathbb{N} \setminus \{0, 1\}, \cdot \rangle$ if for any integer k , both $a_1 \neq a_2^k$ and $a_2 \neq a_1^k$. It is straightforward to see that any two distinct prime numbers a_1 and a_2 greater than 1, are pairwise independent, which immediately implies that the monotone group $\langle \mathbb{Q}, \mathbb{N} \setminus \{0, 1\}, \cdot, \leq \rangle$ is pairwise independent.

Note finally that n -wise independence is *not* a generalization of pairwise independence since it imposes constraints on the integers k_i , $1 \leq i \leq n$.

B Proof that Every Monotone Group is Pairwise Independent

Fix any two distinct integers $l_1, l_2 \in \mathbb{N} \setminus \{0\}$ that are pairwise independent in $\langle \mathbb{N} \setminus \{0\}, + \rangle$. (Since $\langle \mathbb{N} \setminus \{0\}, + \rangle$ is pairwise independent, such integers exist.) Consider now any monotone group $\langle \mathbb{F}, \mathbb{M}, \oplus, \preceq \rangle$. Fix any element $a \in \mathbb{M}$. We will prove that the elements $\bigoplus_{l_1} a$ and $\bigoplus_{l_2} a$ are pairwise independent in $\langle \mathbb{M}, \oplus \rangle$. (Clearly, by monotonicity under composition, these elements are distinct.)

Assume, by way of contradiction, that the elements $\bigoplus_{l_1} a$ and $\bigoplus_{l_2} a$ are *not* pairwise independent in $\langle \mathbb{M}, \oplus \rangle$. Thus, there exists some integer k such that either $\bigoplus_{l_1} a = \bigoplus_k (\bigoplus_{l_2} a)$ or $\bigoplus_{l_2} a = \bigoplus_k (\bigoplus_{l_1} a)$. Assume, without loss of generality, that $\bigoplus_{l_1} a = \bigoplus_k (\bigoplus_{l_2} a)$. Since \oplus is associative, this implies that $\bigoplus_{l_1} a = \bigoplus_{k \cdot l_2} a$. Multiplying right by $\bigoplus_{-k \cdot l_2} a$, we obtain that $(\bigoplus_{l_1} a) \oplus (\bigoplus_{-k \cdot l_2} a) = (\bigoplus_{k \cdot l_2} a) \oplus (\bigoplus_{-k \cdot l_2} a)$. Since \oplus is associative, this implies that $\bigoplus_{l_1 - k \cdot l_2} a = \bigoplus_{k \cdot l_2 - k \cdot l_2} a$, or $\bigoplus_{l_1 - k \cdot l_2} a = e$. Thus, it follows that $l_1 - k \cdot l_2 = 0$ or $l_1 = kl_2$. Thus, the integers l_1 and l_2 are not pairwise independent in $\langle \mathbb{N} \setminus \{0\}, + \rangle$. A contradiction.

C Proof that Some Specific Group is n -Wise Independent

We prove that for any integer $n \geq 2$, the monotone group $\langle \mathbb{Z}, \mathbb{N} \setminus \{0\}, +, \leq \rangle$ is n -wise independent.

Fix any integer $\ell \geq 0$. It suffices to prove that the n integers $2^\ell, 2^{\ell+2}, \dots, 2^{\ell+2(n-1)} \in \mathbb{N} \setminus \{0\}$ are n -wise independent in $\langle \mathbb{N} \setminus \{0\}, + \rangle$.

The proof is by induction on n . For the basis case where $n = 2$, consider the integers 2^ℓ and $2^{\ell+2}$, and fix any pair of integers $k_1, k_2 \in \{-1, 0, 1, 2\}$ that are not both simultaneously zero. Clearly, $k_1 2^\ell + k_2 2^{\ell+2} = 2^\ell(k_1 + 4k_2)$, which can be zero only if $k_1 = k_2 = 0$. This completes the proof of the basis case.

Assume inductively that for all integers $n' < n$, the n' integers $2^\ell, 2^{\ell+2}, \dots, 2^{\ell+2(n'-1)}$ are n' -wise independent in $\langle \mathbb{N} \setminus \{0\}, + \rangle$.

For the induction step, we will show that the n integers $2^\ell, 2^{\ell+2}, \dots, 2^{\ell+2(n-1)}$ are n -wise independent in $\langle \mathbb{N} \setminus \{0\}, + \rangle$. Assume, by way of contradiction, that they are not. Thus, there exist n integers $k_1, k_2, \dots, k_n \in \{-1, 0, 1, 2\}$ that are not all simultaneously zero, such that $\sum_{i=1}^n k_i 2^{\ell+2(i-1)} = 0$. We proceed by case analysis on the value of $k_n \in \{-1, 0, 1, 2\}$.

- Assume first that $k_n = -1$. Then, $\sum_{i=1}^{n-1} k_i 2^{\ell+2(i-1)} - 2^{\ell+2(n-1)} = 0$, or $\sum_{i=1}^{n-1} k_i 2^{\ell+2(i-1)} = 2^{\ell+2(n-1)}$,

or $\sum_{i=1}^{n-1} k_i 2^{2(i-1)} = 2^{2(n-1)}$. However, since $k_i \leq 2$ for all indices i , $1 \leq i \leq n-1$,

$$\begin{aligned} \sum_{i=1}^{n-1} k_i 2^{2(i-1)} &\leq 2 \sum_{i=1}^{n-1} 2^{2(i-1)} \\ &< 2 \cdot \sum_{i=0}^{2n-4} 2^i \\ &= 2(2^{2n-3} - 1) < 2^{2n-2} = 2^{2(n-1)}, \end{aligned}$$

a contradiction.

- Assume now that $k_n = 0$. Then, $\sum_{i=1}^{n-1} k_i 2^{\ell+2(i-1)} = 0$. Since the integers k_1, k_2, \dots, k_n are not all simultaneously zero while $k_n = 0$, it follows that the integers k_1, k_2, \dots, k_{n-1} are not all simultaneously zero. This implies that the $n-1$ integers $2^\ell, 2^{\ell+2}, \dots, 2^{\ell+2(n-2)}$ are $(n-1)$ -wise independent in $\langle \mathbb{N} \setminus \{0\}, + \rangle$, which contradicts the induction hypothesis.
- Assume finally that $k_n \in \{1, 2\}$. Then, $\sum_{i=1}^{n-1} k_i 2^{\ell+2(i-1)} + k_n \cdot 2^{\ell+2(n-1)} = 0$, or, equivalently, $-\sum_{i=1}^{n-1} k_i 2^{\ell+2(i-1)} = k_n \cdot 2^{\ell+2(n-1)}$, or $-\sum_{i=1}^{n-1} k_i 2^{2(i-1)} = k_n \cdot 2^{2(n-1)}$. However, since $k_i \geq -1$ for all indices i , $1 \leq i \leq n-1$,

$$\begin{aligned} -\sum_{i=1}^{n-1} k_i 2^{2(i-1)} &\leq \sum_{i=1}^{n-1} 2^{2(i-1)} \\ &< \sum_{i=0}^{2n-4} 2^i \\ &= 2^{2n-3} - 1 < 2^{2n-2} = 2^{2(n-1)} \leq k_n \cdot 2^{2(n-1)}, \end{aligned}$$

a contradiction.

Since we obtained a contradiction in all possible cases, the proof is now complete.

D Proof that Every Monotone Group is n -Wise Independent

Fix any n distinct integers $l_1, l_2, \dots, l_n \in \mathbb{N} \setminus \{0\}$ that are n -wise independent in $\langle \mathbb{N} \setminus \{0\}, + \rangle$. (Since every monotone group is n -wise independent, such integers exist.) Consider now any monotone group $\langle \Gamma, \mathbb{M}, \oplus, \preceq \rangle$. Fix any element $a \in \mathbb{M}$. We will prove that the n elements $\bigoplus_{l_1} a, \bigoplus_{l_2} a, \dots, \bigoplus_{l_n} a$ of \mathbb{M} are n -wise independent in $\langle \mathbb{M}, \oplus \rangle$. (Clearly, by monotonicity under composition, these elements are distinct.) Assume, by way of contradiction, that the elements $\bigoplus_{l_1} a, \bigoplus_{l_2} a, \dots, \bigoplus_{l_n} a$ are *not* n -wise independent in $\langle \mathbb{M}, \oplus \rangle$. Thus, there exist n integers $k_1, k_2, \dots, k_n \in \{-1, 0, 1, 2\}$, that are not all simultaneously zero, such that $\biguplus_{i=1}^n (\bigoplus_{k_i} (\bigoplus_{l_i} a)) = e$. Since \oplus is associative, this implies that $\biguplus_{i=1}^n (\bigoplus_{k_i \cdot l_i} a) = e$, or, by the definition of the summation operation, $\bigoplus_{\sum_{i=1}^n k_i \cdot l_i} a = e$. Hence, it follows that $\sum_{i=1}^n k_i \cdot l_i = 0$. Since the integers k_i , $1 \leq i \leq n$, are all in $\{-1, 0, 1, 2\}$ and not all simultaneously zero, this implies that the integers l_1, l_2, \dots, l_n are n -wise independent in $\langle \mathbb{N} \setminus \{0\}, + \rangle$. A contradiction.

E Proof of Unique Serialization Lemma

Assume, by way of contradiction, that there are two distinct valid serializations

$$S^{(1)}(\alpha) = op^{(1.1)}, op^{(1.2)}, op^{(1.3)}, \dots$$

and

$$S^{(2)}(\alpha) = op^{(2.1)}, op^{(2.2)}, op^{(2.3)}, \dots$$

of execution α . Since $S^{(1)}(\alpha)$ and $S^{(2)}(\alpha)$ are distinct, there exists a least index $k \geq 1$ such that $op^{(1,k)}$ is different from $op^{(2,k)}$. Assume, without loss of generality, that $op^{(1,k)}$ appears at position $l > k$ in the serialization $S^{(2)}(\alpha)$ (that is, $op^{(1,k)} = op^{(2,l)}$). Notice that for each $i < k$, $op^{(1,i)} = op^{(2,i)}$.

We examine two cases: $k = 1$ and $k > 1$. First consider the case $k = 1$. Since $S^{(1)}(\alpha)$ is a valid serialization of α , we have that $\text{Out}(op^{(1,k)}) = e$, and since $S^{(2)}(\alpha)$ is a valid serialization of α , $\text{Out}(op^{(2,l)}) = \text{Out}(op^{(2,l-1)}) \oplus \text{In}(op^{(2,l-1)}) \neq e$ (this follows from the identity lower bound for \mathbb{M}). Which implies that $\text{Out}(op^{(1,k)}) \neq \text{Out}(op^{(2,l)})$, a contradiction.

Now we consider the case $k > 1$. Since $S^{(1)}(\alpha)$ is a valid serialization of α ,

$$\text{Out}(op^{(1,k)}) = \text{Out}(op^{(1,k-1)}) \oplus \text{In}(op^{(1,k-1)}).$$

Since $S^{(2)}(\alpha)$ is a valid serialization of α ,

$$\text{Out}(op^{(2,l)}) = \text{Out}(op^{(2,k-1)}) \oplus \text{In}(op^{(2,k-1)}) \oplus \dots \oplus \text{In}(op^{(2,l-2)}) \oplus \text{In}(op^{(2,l-1)}).$$

Since $\text{Out}(op^{(1,k)}) = \text{Out}(op^{(2,l)})$,

$$\begin{aligned} \text{Out}(op^{(1,k-1)}) \oplus \text{In}(op^{(1,k-1)}) &= \\ \text{Out}(op^{(2,k-1)}) \oplus \text{In}(op^{(2,k-1)}) \oplus \dots \oplus \text{In}(op^{(2,l-2)}) \oplus \text{In}(op^{(2,l-1)}). \end{aligned}$$

Since $\text{Out}(op^{(1,k-1)}) = \text{Out}(op^{(2,k-1)})$ and $\text{In}(op^{(1,k-1)}) = \text{In}(op^{(2,k-1)})$,

$$\begin{aligned} \text{Out}(op^{(2,k-1)}) \oplus \text{In}(op^{(2,k-1)}) &= \\ \text{Out}(op^{(2,k-1)}) \oplus \text{In}(op^{(2,k-1)}) \oplus \dots \oplus \text{In}(op^{(2,l-2)}) \oplus \text{In}(op^{(2,l-1)}). \end{aligned}$$

Let $\text{Out}(op^{(2,k-1)}) \oplus \text{In}(op^{(2,k-1)}) = a$, we obtain

$$a = a \oplus \text{In}(op^{(2,k)}) \oplus \dots \oplus \text{In}(op^{(2,l-2)}) \oplus \text{In}(op^{(2,l-1)}).$$

Hence,

$$a^{-1} \oplus a = a^{-1} \oplus a \oplus \text{In}(op^{(2,k)}) \oplus \dots \oplus \text{In}(op^{(2,l-2)}) \oplus \text{In}(op^{(2,l-1)}).$$

From the associativity of operation \oplus we obtain $e = e \oplus \text{In}(op^{(2,k)}) \oplus \dots \oplus \text{In}(op^{(2,l-1)})$. Thus, $e = \text{In}(op^{(2,k)}) \oplus \dots \oplus \text{In}(op^{(2,l-1)})$. By the identity lower bound for \mathbb{M} , $e \prec \text{In}(op^{(2,k)}) \oplus \dots \oplus \text{In}(op^{(2,l-1)})$. A contradiction.

F The Monotone Sequential Consistency Lemma and its Proof

To provide the reader with useful intuition for the Monotone Linearizability Lemma and its proof, we first prove a corresponding *Monotone Sequential Consistency Lemma*, which applies to a somehow simpler context.

We consider a system \mathbf{P} that implements a monotone group $\langle \mathbb{I}, \mathbb{M}, \oplus, \preceq \rangle$. Say that a process p_i is *sequentially consistent in execution* α [14] if the (unique) valid serialization $S(\alpha)$ extends $\xrightarrow{\alpha}_i$; that is, for any pair of operations $op_i^{(1)}$ and $op_i^{(2)}$ such that $op_i^{(1)} \xrightarrow{\alpha}_i op_i^{(2)}$, $op_i^{(1)} \xrightarrow{S(\alpha)} op_i^{(2)}$. Since \mathbf{P} implements the monotone group $\langle \mathbb{I}, \mathbb{M}, \oplus, \preceq \rangle$, for any two operations $op_i^{(1)}$ and $op_i^{(2)}$ such that $op_i^{(1)} \xrightarrow{S(\alpha)} op_i^{(2)}$, $\text{Out}(op_i^{(1)}) \prec \text{Out}(op_i^{(2)})$. Thus, it follows that for any process p_i that is sequentially consistent in execution α , for any pair of operations $op_i^{(1)}$ and $op_i^{(2)}$ such that $op_i^{(1)} \xrightarrow{\alpha} op_i^{(2)}$, $\text{Out}(op_i^{(1)}) \prec \text{Out}(op_i^{(2)})$. Say that an execution α is *sequentially consistent* [14] if every process p_i is sequentially consistent in execution α . Say that operation $op_i^{(1)}$ at process p_i in execution α is *sequentially inconsistent in execution* α if there is another operation $op_i^{(2)}$ at the same process in execution α such that $op_i^{(2)} \xrightarrow{\alpha} op_i^{(1)}$ while $\text{Out}(op_i^{(1)}) \prec \text{Out}(op_i^{(2)})$. Say that operation $op_i^{(1)}$ at process p_i in execution α is *sequentially consistent in execution* α if it is not sequentially inconsistent in execution α . Clearly, process p_i is sequentially consistent in execution α if every operation op_i at process p_i in execution α is sequentially consistent in it.

Proposition F.1 (Monotone Sequential Consistency Lemma) *for any execution α of system \mathbf{P} in which process p_1 issues only operations of type a_1 , while any other process p_i , $i \neq 1$, issues only operations of type a_2 , p_1 is sequentially consistent in execution α .*

Proof: Assume, by way of contradiction, that process p_1 is not sequentially consistent in execution α . So, there is at least one operation at p_1 that is sequentially inconsistent in execution α ; consider the earliest such operation $op_1^{(1)}$. It follows that for the operation $op_1^{(2)}$ that immediately precedes $op_1^{(1)}$ in α , $\text{Out}(op_1^{(1)}) \prec \text{Out}(op_1^{(2)})$.

Consider the finite prefix β_1 of execution α that ends with the response for operation $op_1^{(1)}$. Consider now a finite execution α_1 that is an extension of β_1 that includes no additional invocations by processes; thus, α_1 contains only responses to invocations that are pending in β_1 . Clearly, all operations whose responses precede or coincide with that of $op_1^{(1)}$ have identical outputs in α and α_1 . This implies, in particular, that $op_1^{(1)}$ is the only one sequentially inconsistent operation at process p_1 in execution α_1 .

Take now the (unique) valid serialization $S(\alpha_1)$ of α_1 . Clearly, $op_1^{(1)}$ precedes $op_1^{(2)}$ in $S(\alpha_1)$. Denote $k \geq 0$ the number of operations at p_1 that precede $op_1^{(2)}$ in execution α_1 . Since $op_1^{(2)}$ is sequentially consistent in execution α_1 , all these operations must precede $op_1^{(2)}$ in $S(\alpha_1)$; recall that all operations at p_1 are of type a_1 . Denote l_1 the number of operations at other processes that precede $op_1^{(2)}$ in the serialization $S(\alpha_1)$; recall that all these operations are of type a_2 . Since $S(\alpha_1)$ is a valid serialization of α_1 , we use the the associativity and commutativity of operation \oplus to obtain that

$$\begin{aligned} \text{Out}\left(op_1^{(2)}\right) &= \bigoplus_k a_1 \oplus \text{In}\left(op_1^{(1)}\right) \oplus \bigoplus_{l_1} a_2 \\ &= \bigoplus_k a_1 \oplus a_1 \oplus \bigoplus_{l_1} a_2. \end{aligned}$$

Consider now the finite prefix β_2 of execution α that ends with the response for operation $op_1^{(2)}$. Consider now a finite execution α_2 that is an extension of β_2 that includes no additional invocations by processes; thus, α_2 contains only responses to invocations that are pending in β_2 . Clearly, all operations whose responses precede or coincide with that of $op_1^{(2)}$ have identical outputs in α and α_2 . This implies, in particular, that there is no sequentially inconsistent operation at process p_1 in execution α_2 .

Take now the (unique) valid serialization $S(\alpha_2)$ of execution α_2 . Denote $k \geq 0$ the number of operations at p_1 that precede $op_1^{(2)}$ in execution α_1 . Since $op_1^{(2)}$ is sequentially consistent in execution α_1 , all k operations at p_1 that precede $op_1^{(2)}$ in execution α , and hence in execution α_2 as well, must also precede $op_1^{(2)}$ in the serialization $S(\alpha_2)$; recall that all operations at p_1 are of type a_1 . Denote l_2 the number of operations at other processes that precede $op_1^{(2)}$ in the serialization $S(\alpha_2)$; recall that all these operations are of type a_2 . Since $S(\alpha_2)$ is a valid serialization of α_2 , we use the the associativity and commutativity of operation \oplus to obtain that

$$\text{Out}\left(op_1^{(2)}\right) = \bigoplus_k a_1 \oplus \bigoplus_{l_2} a_2.$$

Recall that all operations whose responses precede or coincide with that of $op_1^{(1)}$ in α have identical outputs in α and α_1 , and that all operations whose responses precede or coincide with that of $op_1^{(2)}$ in α have identical outputs in α and α_2 . In particular, these hold for the operation $op_1^{(2)}$, which implies that it has identical outputs in α_1 and α_2 . It follows that

$$\bigoplus_k a_1 \oplus a_1 \oplus \bigoplus_{l_1} a_2 = \bigoplus_k a_1 \oplus \bigoplus_{l_2} a_2.$$

Multiplying both sides by $(\bigoplus_k a_1)^{-1}$ from the left, it follows that $a_1 \oplus \bigoplus_{l_1} a_2 = \bigoplus_{l_2} a_2$. Multiplying both sides by $\bigoplus_{-l_1} a_2$ from the right, we obtain that $a_1 = (\bigoplus_{-l_1} a_2) \oplus (\bigoplus_{l_2} a_2) = (\bigoplus_{l_1} a_2^{-1}) \oplus (\bigoplus_{l_2} a_2)$. We proceed by case analysis on how l_1 and l_2 compare to each other.

- Assume first that $l_1 = l_2$. Then, clearly, $a_1 = e$, a contradiction.
- Assume now that $l_1 < l_2$. Then, clearly, $a_1 = \bigoplus_{l_2-l_1} a_2$. It follows that a_1 and a_2 are not pairwise independent, a contradiction.
- Assume finally that $l_1 > l_2$. Then, clearly, $a_1 = \bigoplus_{l_1-l_2} a_2^{-1} = \bigoplus_{l_2-l_1} a_2$. It follows that a_1 and a_2 are not pairwise independent, a contradiction.

Since we obtained a contradiction in all possible cases, the proof is now complete. \blacksquare

G Proof of the Monotone Linearizability Lemma

Assume, by way of contradiction, that α is not linearizable. So, there is at least one operation that is non-linearizable in α . Consider the non-linearizable operation op_k at process p_k with *earliest* response in execution α (among all non-linearizable operations in α). Since op_k is non-linearizable, there exists an operation op_l at some process l such that $op_l \xrightarrow{\alpha} op_k$ while $\text{Out}(op_k) \prec \text{Out}(op_l)$; fix op_l to be the one with *latest* response in execution α .

Consider the finite prefix β_1 of execution α that ends with the response for operation op_k . Consider now a finite execution α_1 , which is an extension of β_1 that includes no additional invocations by processes; thus, α_1 contains, in addition to those in β_1 , only responses to invocations that are pending in β_1 . Clearly, all operations whose responses precede or coincide with that of op_k have identical outputs in α and α_1 . This implies, in particular, that the non-linearizable operations in α_1 are op_k and possibly operations with responses following that of op_k (but at most one such operation per process).

Take now the (unique) valid serialization $S(\alpha_1)$ of α_1 . Clearly, op_k precedes op_l in $S(\alpha_1)$. In $S(\alpha_1)$, operation op_k must appear before op_l . For each process p_i , where $1 \leq i \leq n$, denote m_i the number of operations at p_i that precede op_l in the serialization $S(\alpha_1)$. Partition these m_i operations into two classes:

- Operations whose response in α_1 precedes or coincides with that of op_l ; (thus, op_l falls into this class); assume there are $m_{i,a}$ of them.
- Operations whose response in α_1 follow that of op_l ; assume there are $m_{i,b}$ of them;

thus, $m_i = m_{i,a} + m_{i,b}$. Since each operation at process p_i is of type a_i , and since $S(\alpha_1)$ is a valid serialization of α_1 , we use the associativity and commutativity of operation \oplus to obtain that

$$\begin{aligned} \text{Out}(op_l) &= \bigoplus_{i=1}^n \bigoplus_{m_{i,a}+m_{i,b}} a_i \\ &= \left(\bigoplus_{i=1}^n \bigoplus_{m_{i,a}} a_i \right) \oplus \left(\bigoplus_{i=1}^n \bigoplus_{m_{i,b}} a_i \right). \end{aligned}$$

We continue to prove a simple claim:

Claim G.1 For each process p_i , $0 \leq m_{i,b} \leq 2$.

Proof: We proceed by case analysis.

1. Assume first that $i \notin \{k, l\}$, and consider any operation op_i whose response follows that of op_l . Hence, either $op_l \xrightarrow{\alpha_1} op_i$ or $op_l \parallel_{\alpha_1} op_i$. We proceed by case analysis.
 - Assume first that $op_l \xrightarrow{\alpha_1} op_i$. Since op_i precedes op_l in the valid serialization $S(\alpha_1)$, $\text{Out}(op_i) \prec \text{Out}(op_l)$. Thus, op_i is non-linearizable in execution α_1 . It follows that the response for op_i follows the one for op_k . Since, by construction, α_1 contains no invocation following the response for op_k , it follows that there can be at most one such operation op_i .
 - Assume now that $op_l \parallel_{\alpha_1} op_i$. Since the response for op_i follows that of op_l , it follows that there can be at most one such operation op_i (since any other following the one such operation with earliest response would satisfy $op_l \xrightarrow{\alpha_1} op_i$).

It follows from the case analysis that $0 \leq m_{i,b} \leq 2$.

2. Assume now that $i = k$, and consider any operation op'_k at process p_k whose response follows that of op_l in α_1 . Hence, either $op_l \xrightarrow{\alpha_1} op'_k$ or $op_l \parallel_{\alpha_1} op'_k$. We proceed by case analysis.
 - Assume first that $op_l \xrightarrow{\alpha_1} op'_k$. Clearly, op_k is one such operation. Any other op'_k different than op_k cannot have its invocation following the response for op_k in α_1 , since α_1 does not contain any invocations following the response for op_k . Hence, op'_k precedes op_k in α_1 . It follows that op'_k is linearizable in execution α_1 . Since $op_l \xrightarrow{\alpha_1} op'_k$, it follows that $\text{Out}(op_l) \prec \text{Out}(op'_k)$. Since $S(\alpha_1)$ is a valid serialization of α_1 , op'_k follows op_l in $S(\alpha_1)$, a contradiction. This implies that no other operation op'_k exists.
 - Assume now that $op_l \parallel_{\alpha_1} op'_k$. Clearly, the invocation for operation op'_k precedes the response for operation op_l , while the response for operation op'_k precedes the invocation for operation op_k (since no two operations at the same process may be simultaneously pending). It follows that there can be at most one such operation op'_k .

It follows from the case analysis above that $1 \leq m_{k,b} \leq 2$.

3. Assume finally that $i = l$, and consider any operation op'_l at process p_l whose response follows that of op_l in α_1 . Since no two operations at the same process may be simultaneously pending, it follows that op_l precedes op'_l in α_1 . Since op'_l precedes op_l in $S(\alpha_1)$, it follows that op'_l is non-linearizable in execution α_1 . Hence, the response for op'_l follows that of op_k in α_1 . Since there are no invocations following the response for op_k in α_1 , it follows that there can be at most one such operation op'_l , and $0 \leq m_{l,b} \leq 1$.

The previous analysis implies that for any process p_i , where $1 \leq i \leq n$, $0 \leq m_{i,b} \leq 2$, as needed. ■

Consider now the finite prefix β_2 of execution α that ends with the response for operation op_l . Consider a finite execution α_2 , which is an extension of β_2 that includes no additional invocations by processes; thus, α_2 contains, in addition to those in β_2 , only responses to invocations that are pending in β_2 . Clearly, all operations whose responses precede or coincide with that of op_l have identical outputs in α and α_2 . Note also that, by construction, there are no non-linearizable operations in α_2 .

Consider the (unique) valid serialization $S(\alpha_2)$ of α_2 . For each process p_i , where $1 \leq i \leq n$, denote m'_i the number of operations at process p_i that precede op_l in the serialization $S(\alpha_2)$. Partition these m'_i operations into two classes:

- (a) Operations whose response in α_2 precedes or coincides with that of op_l ; (thus, op_l falls into this class); assume there are $m'_{i,a}$ of them.
- (b) Operations whose response in α_2 follow that of op_l ; assume there are $m'_{i,b}$ of them;

thus, $m'_i = m'_{i,a} + m'_{i,b}$. Since each operation at process p_i is of type a_i , and since $S(\alpha_1)$ is a valid serialization of α_1 , we use the associativity and commutativity of operation \oplus to obtain that

$$\begin{aligned} \text{Out}(op_l) &= \bigoplus_{i=1}^n \bigoplus_{m'_{i,a} + m'_{i,b}} a_i \\ &= \left(\bigoplus_{i=1}^n \bigoplus_{m'_{i,a}} a_i \right) \oplus \left(\bigoplus_{i=1}^n \bigoplus_{m'_{i,b}} a_i \right). \end{aligned}$$

Clearly, all operations whose responses precede or coincide with that of op_l have the same outputs in α and α_1 , and in α and α_2 , respectively. It follows that all such operations have the same outputs in α_1 and α_2 . Consider any such operation op . If op precedes op_l in serialization $S(\alpha_1)$ then it must be that $\text{Out}(op) \prec \text{Out}(op_l)$. Subsequently, it must be that op precedes op_l in serialization $S(\alpha_2)$ too, since otherwise the valid composition condition of the valid serialization for a monotone group would be violated. For the

same reason, if op does not precede op_l in serialization $S(\alpha_1)$ then it must be that op does not precede op_l in serialization $S(\alpha_2)$ either. This implies that for all processes p_i , $1 \leq i \leq n$, $m_{i,a} = m'_{i,a}$.

Since operation op_l has the same output in executions α_1 and α_2 , it follows that

$$\left(\biguplus_{i=1}^n \bigoplus_{m_{i,a}} a_i \right) \oplus \left(\biguplus_{i=1}^n \bigoplus_{m_{i,b}} a_i \right) = \left(\biguplus_{i=1}^n \bigoplus_{m'_{i,a}} a_i \right) \oplus \left(\biguplus_{i=1}^n \bigoplus_{m'_{i,b}} a_i \right).$$

Since for each process p_i , $1 \leq i \leq n$, $m_{i,a} = m'_{i,a}$, it follows that

$$\left(\biguplus_{i=1}^n \bigoplus_{m_{i,a}} a_i \right) \oplus \left(\biguplus_{i=1}^n \bigoplus_{m_{i,b}} a_i \right) = \left(\biguplus_{i=1}^n \bigoplus_{m_{i,a}} a_i \right) \oplus \left(\biguplus_{i=1}^n \bigoplus_{m'_{i,b}} a_i \right).$$

Multiplying both sides from the left by $\left(\biguplus_{i=1}^n \bigoplus_{m_{i,a}} a_i \right)^{-1}$, we obtain that

$$\left(\biguplus_{i=1}^n \bigoplus_{m_{i,a}} a_i \right)^{-1} \oplus \left(\biguplus_{i=1}^n \bigoplus_{m_{i,a}} a_i \right) \oplus \left(\biguplus_{i=1}^n \bigoplus_{m_{i,b}} a_i \right) = \left(\biguplus_{i=1}^n \bigoplus_{m_{i,a}} a_i \right)^{-1} \oplus \left(\biguplus_{i=1}^n \bigoplus_{m_{i,a}} a_i \right) \oplus \left(\biguplus_{i=1}^n \bigoplus_{m'_{i,b}} a_i \right),$$

or, using the associativity of operation \oplus , that

$$\biguplus_{i=1}^n \bigoplus_{m_{i,b}} a_i = \biguplus_{i=1}^n \bigoplus_{m'_{i,b}} a_i.$$

Multiplying both sides from the right by $\biguplus_{i=1}^n \bigoplus_{-m'_{i,b}} a_i$, we obtain that

$$\left(\biguplus_{i=1}^n \bigoplus_{m_{i,b}} a_i \right) \oplus \left(\biguplus_{i=1}^n \bigoplus_{-m'_{i,b}} a_i \right) = \left(\biguplus_{i=1}^n \bigoplus_{m'_{i,b}} a_i \right) \oplus \left(\biguplus_{i=1}^n \bigoplus_{-m'_{i,b}} a_i \right).$$

By the associativity of the operation \oplus , this implies that

$$\biguplus_{i=1}^n \bigoplus_{m_{i,b} - m'_{i,b}} a_i = e.$$

We continue to prove:

Claim G.2 For all processes p_i , where $1 \leq i \leq n$, $0 \leq m'_{i,b} \leq 1$.

Proof: Consider first any operation op_i at process p_i , where $i \neq l$, whose response follows that of op_l . By construction of α_2 , no other operation at p_i follows op_i in α_2 , so that $m'_{i,b} \leq 1$ for $i \neq l$. For $i = l$, by construction of α_2 , no operation at p_l follows op_l in α_1 , so that $m'_{l,b} = 0$. It follows that for all processes p_i , $1 \leq i \leq n$, $0 \leq m'_{i,b} \leq 1$, as needed. \blacksquare

By Claims G.1 and G.2, it immediately follows:

Claim G.3 For all processes p_i , $-1 \leq m_{i,b} - m'_{i,b} \leq 2$.

Clearly, Claim G.3 is not sufficient to apply to $\biguplus_{i=1}^n \bigoplus_{m_{i,b} - m'_{i,b}} a_i = e$ in order to conclude that the elements a_i , $1 \leq i \leq n$ are n -wise independent (and derive a contradiction), since it needs also be established that the differences $m_{i,b} - m'_{i,b}$, where $1 \leq i \leq n$, are not all simultaneously zero. We continue to prove this.

Claim G.4 Not all differences $m_{i,b} - m'_{i,b}$, where $1 \leq i \leq n$, are simultaneously zero.

Proof: It suffices to prove that the difference $m_{k,b} - m'_{k,b}$ never vanishes. Recall, from the proof of Claim G.1, that $m_{k,b} = 1$. Hence, it suffices to show that $m'_{k,b} = 0$. We proceed by case analysis.

- Assume first that there is no operation op'_k at process p_k in execution α_1 such that $op_l \parallel_{\alpha_1} op'_k$, and the response of op'_k follows that of op_l . By the way α_1 and α_2 have been constructed, it follows that there is neither an operation op'_k at process p_k in execution α_2 such that $op_l \parallel_{\alpha_2} op'_k$, and the response of op'_k follows that of op_l . Thus, in this case, $m'_{k,b} = 0$.
- Assume now that there is (at most one) operation op'_k at process p_k such that $op_l \parallel_{\alpha_2} op'_k$, and the response for op'_k follows that of op_l . We will show that operation op'_k does not exist. Assume for contradiction that operation op'_k exists. Then, in serialization $S(\alpha_1)$ operation op'_k does not precede op_l , since otherwise operation op'_k would be of class (b) and since op_k is of class (b) we would have $m_{k,b} = 2$. This implies that in α_1 , $\text{Out}(op_k) \prec \text{Out}(op'_k)$. Subsequently, since the responses of operations op_l and op'_k precede the response of operation and op_k , it must be that in execution α , $\text{Out}(op_k) \prec \text{Out}(op'_k)$ and the response of op'_k follows that of op_l . This implies that op_l is not the latest operation in α such that $\text{Out}(op_k) \prec \text{Out}(op_l)$, a contradiction.

Thus, $m'_{k,b} = 0$ and $m_{k,b} - m'_{k,b} = 1 \neq 0$, as needed to complete the proof. ■

By Claim G.4, it follows from the equation $\bigoplus_{i=1}^n \bigoplus_{m_{i,b}-m'_{i,b}} a_i = e$ that the n elements a_1, a_2, \dots, a_n are not n -wise independent. A contradiction.