Complexity Tradeoffs for Read and Update Operations*

Danny Hendler
Department of Computer-Science
Ben-Gurion University
hendlerd@cs.bgu.ac.il

Vitaly Khait
Department of Computer-Science
Ben-Gurion University
khvitaly@gmail.com

ABSTRACT

Recent work established that some restricted-use objects, such as max registers, counters and atomic snapshots, admit polylogarithmic step-complexity wait-free implementations using only reads and writes: when only polynomially-many updates are allowed, reading the object (by performing a `ReadMax`, `CounterRead` or `Scan` operation, depending on the object's type) incurs $O(\log N)$ steps (where N is the number of processes), which was shown to be optimal.

But what about the step-complexity of update operations? With these implementations, updating the object's state (by performing a `WriteMax`, `CounterIncrement` or `Update` operation, depending on the object's type) requires $\Omega(\log N)$ steps. The question that we address in this work is the following: are there read-optimal implementations of these restricted-use objects for which the asymptotic step-complexity of update operations is sub-logarithmic?

We present tradeoffs between the step-complexity of read and update operations on these objects, establishing that updating a read-optimal counter or snapshot incurs $\Omega(\log N)$ steps. These tradeoffs hold also if compare-and-swap (CAS) operations may be used, in addition to reads and writes.

We also derive a tradeoff between the step-complexities of read and update operations of M-bounded max registers: if the step-complexity of the `ReadMax` operation is $O\big(f(min(N,M))\big)$, then the step-complexity of the `WriteMax` operation is $\Omega(\log \frac{\log min(N,M)}{\log f(min(N,M))})$. It follows from this tradeoff that the step-complexity of `WriteMax` in any read-optimal implementation of a max register from read, write and CAS is $\Omega\big(\log\log min(N,M)\big)$. On the positive side, we present a wait-free implementation of an M-bounded max register from read, write and CAS for which the step complexities of `ReadMax` and `WriteMax` operations are $O(1)$ and $O\big(\log min(N,M)\big)$, respectively.

*Partially supported by the Israel Science Foundation (grant number 1227/10) and by the Lynne and William Frankel Center for Computing Science at Ben-Gurion University.

PODC'14, July 15–18, 2014, Paris, France.

http://dx.doi.org/10.1145/2611462.2611472.

Categories and Subject Descriptors

D.1.3 [**Software**]: Programming Techniques—*Concurrent Programming*

Keywords

Counter, max register, snapshot, restricted-use objects.

1. INTRODUCTION

Concurrent objects are often constructed only for a restricted use. Their use can be restricted either by limiting the number of operations applied to them (e.g., an *atomic snapshot* object that only supports a limited number of `Update` operations), or by passing bounded arguments to the operations applied to them (e.g., a *max register* that supports only writes of values below some threshold).

Recent work established that some restricted-use objects admit polylogarithmic step-complexity wait-free [10] implementations using only reads and writes. Aspnes, Attiya and Censor-Hillel [2] presented a wait-free implementation of an M-bounded max register that supports a `WriteMax` operation that writes a value to the max register, and a `ReadMax` operation that returns the largest value previously written. Both these operations have step-complexity of $O(\log M)$. By using max register as a building block, they constructed an efficient wait-free counter. `CounterRead` operations on the counter incur a number of steps logarithmic in N, the number of processes sharing the implementation. If the counter is required to support only a limited number of `CounterIncrement` operations (polynomial in N), then the step complexity of `CounterIncrement` operations is $O(\log^2 N)$. Restricted-use max registers and counters have already been used for devising efficient randomized consensus [5] and mutual exclusion algorithms [7].

More recently, Aspnes et al. [3] presented an implementation of restricted-use atomic snapshots. For polynomially-many updates, the step-complexities of the `Scan` and `Update` operations in their implementation are $O(\log N)$ and $O(\log^3 N)$, respectively.

Are the step-complexities of these implementations asymptotically optimal? For read operations, it was shown that they are. Assuming (as we do in the rest of this paper) polynomially-many updates, Aspnes et al. [2] proved an $\Omega(\log M)$ step lower bound on `ReadMax` operations for obstruction-free implementations of M-bounded max registers. They also proved an $\Omega(\log N)$ step-complexity lower bound on `CounterRead` operations for obstruction-free counter implementations. This result was later generalized by Aspnes et al. [4] for a class of objects that includes atomic snapshots, establishing an $\Omega(\log N)$ step-complexity lower bound on `Scan` operations for obstruction-free implementations of snapshots.

But what about the step-complexity of update operations on these objects? The `WriteMax` operation of the max register algorithm presented in [2] has logarithmic step-complexity, and the counter `CounterIncrement` and snapshot `Update` operations of the algorithms presented in [2, 3] have poly-logarithmic step-complexities. Can we do better? *Are there read-optimal implementations of these restricted-use objects for which the asymptotic step-complexity of update operations is sub-logarithmic or even constant?* This is the question that we address in this paper.

Our Contributions:
We prove the following tradeoff for counters and atomic snapshots: if the step-complexity of the read operation is $O\big(f(N)\big)$, then the step-complexity of the update operation is $\Omega(\log \frac{N}{f(N)})$. The tradeoff holds for obstruction-free implementations, even if CAS may be used in addition to reads and writes. Setting $f(N) = \log N$ establishes that for any read-optimal implementation of a counter or snapshot object from reads and writes, the step-complexity of update operations is $\Omega(\log N)$.

For M-bounded max registers, we were only able to obtain a weaker tradeoff: if the step-complexity of the `ReadMax` operation is $O\big(f(min(N, M))\big)$, then the step-complexity of the `WriteMax` operation is $\Omega(\log \frac{\log min(N,M)}{\log f(min(N,M))})$. It follows from this tradeoff that the step-complexity of `WriteMax` in any read-optimal implementation of a max register from read, write and CAS is $\Omega\big(\log\log min(N, M)\big)$. On the positive side, we present a wait-free implementation of an M-bounded max register from read, write and CAS for which the step complexities of `ReadMax` and `WriteMax` operations are $O(1)$ and $O\big(\log min(N, M)\big)$, respectively.

2. PRELIMINARIES

We consider a standard model of an asynchronous shared memory system, in which processes communicate by applying operations to shared objects. An object is characterized by a domain of possible values and by a set of *operations* that provide the only means to manipulate it. An *implementation* of an object shared by a set $\mathbf{P}= \{p_1, \cdots, p_N\}$ of N processes provides a specific data-representation for the object from a set $\mathbf{B}$ of shared *base objects*, each of which is assigned an initial value; the implementation also provides algorithms for each process in $\mathbf{P}$ to apply each operation to the object being implemented.

A *wait-free* implementation of a concurrent object guarantees that any process can complete an operation in a finite number of its own steps. An *obstruction-free* [11] implementation guarantees only that if a process eventually runs by itself then it completes its operation within a finite number of its own steps. Each *step* consists of some local computation and one shared memory *event*, which is a *primitive operation* (or simply *primitive*) applied atomically to an object in $\mathbf{B}$ that may return a response value. We say that the event *accesses* the object and that it *applies* the primitive to it. We say that an event is *trivial*, if it does not change the value of the base object to which it is applied.

An *execution fragment* is a (finite or infinite) sequence of events. An *execution* is an execution fragment that starts from the *initial configuration*, resulting when processes apply operations to the implemented object as they execute the implementation's algorithm. We let $\perp$ denote the empty execution. For any finite execution fragment E and any execution fragment E', the execution fragment EE' denotes the concatenation of E and E'. Let E and F be two executions. We say that F is *an extension of* E if $F = EE'$ for some execution fragment E'. We say that executions E, E' are *indistinguishable* to process p, if p performs the same events and receives the same responses to these events in both executions. For an execution E and a set of processes P, we let E^{-P} denote the sequence of events obtained by removing all the events issued by the processes of P from E.

If a process has not completed its operation, it has exactly one *enabled* event, which is the next event it will issue, as specified by the algorithm it is using to apply its operation to the implemented object. We say that a process p is *active* after E if p has not completed its operation in E. Operation Φ_1 *precedes* operation Φ_2 in execution E, if Φ_1 completes in E before the first event of Φ_2 has been issued in E.

The *compare-and-swap* (CAS) operation is defined as follows: *CAS(v,expected,new)* changes the value of variable v to *new* only if its value just before CAS is applied is *expected*; in this case, the *CAS* operation returns *true* and we say it is *successful.* Otherwise, CAS does not change the value of v and returns *false*; in this case, we say that the *CAS* was *unsuccessful.* We assume that base objects support only the read, write, and CAS primitives.

A *max register* supports a `WriteMax`(v) operation, which writes the value v to the object, and a `ReadMax` operation; in its sequential specification, `ReadMax` returns the maximum value written by a `WriteMax` operation instance preceding it. In the bounded version of these objects, the max register is only required to satisfy its specification if its associated value does not exceed a certain threshold M. A *counter* object supports a `CounterIncrement` operation and a `CounterRead` operation; the sequential specification of a counter requires that a `CounterRead` operation instance returns the number of `CounterIncrement` operation instances that precede it. An (atomic) *single-writer snapshot* object consist of an array of N *segments.* An `Update` operation by process p_i atomically sets the value of segment i. The `Scan` operation atomically reads the values of all segments.

3. TRADEOFF FOR COUNTERS AND SNAPSHOT OBJECTS

In this section, we prove the following theorem.

THEOREM 1. *Let I be an N-process obstruction-free implementation of a counter from the* read*,* write *and* CAS *primitives. If the step-complexity of `CounterRead` is $O\left(f(N)\right)$, then the step-complexity of `CounterIncrement` is $\Omega\left(\log \frac{N}{f(N)}\right)$.*

By way of reduction, we show a similar result for single-writer snapshot objects. A key idea underlying our proofs, previously used by [13], is that the rate at which processes become aware of the existence of others is not too rapid. Our proofs extend a technique presented in [6], based on the following definitions which allow quantifying the extent of inter-process communication.

DEFINITION 1. *Let e be an event applied by process p to a base object o in an execution E, where $E = E_1 e E_2$. We say* *that e is* invisible in E, *if either the value of o is not changed by e or if $E_2 = E'e'E''$, e' is a write event to o, p does not take steps in E', and no event in E' is applied to o.*

Informally, an *invisible* event is an event by some process that cannot be observed by other processes. If an event e applied to some object o in an execution E is not *invisible*, we say that e *is visible in* E *on* o.

We next capture the extent by which processes are aware of the participation of other processes in an execution. Intuitively, a process p is aware of the participation of another process q in an execution if there is (either direct or indirect) information flow from q to p in that execution via shared memory. The following definitions formalize this notion.

DEFINITION 2. *Let e_q be an event by process q in an execution E, which applies a write or a CAS primitive to a base object o. We say that an event e_p in E by process p is* aware *of e_q, if e_p accesses o and at least one of the following holds: 1) There is a prefix E' of E such that e_q is visible on o in E' and e_p is a read or CAS event that follows e_q in E', or 2) there is an event e_r that is aware of e_q in E and e_p is aware of e_r in E. If an event e_p of process p is aware of an event e_q of process q in E, we say that p is* aware *of e_q and that e_p is* aware *of q in E.*

DEFINITION 3. *Process p is* aware *of process q after an execution E if either $p = q$ or if p is aware of an event of q in E. The* awareness set *of p after E, denoted $AW(p, E)$, is the set of processes that p is aware of after E.*

DEFINITION 4. *Let E be an execution, o be a base object, and q be a process. We say that o* is familiar *with q after E if there is an event e, visible on o in E, such that $E = E_1eE_2$ and e is an application of a write or a CAS primitive to o by some process r such that $q \in AW(r, E_1e)$. The* familiarity set *of o after E, denoted $F(o, E)$, contains all processes that o is familiar with after E.*

A more general version of the following lemma for implementations that use *read*, *write* and *k-CAS* primitives appears in [6]. For the sake of presentation completeness, we provide here a simpler proof for our model.

LEMMA 1. *Let E be an execution. Let $\mathcal{M}(E) = \max_{p,o}(\{|AW(p, E)| | p \in P\} \cup \{|F(o, E)| | o \in B\})$ be the maximum size of all familiarity and awareness sets after E. Let S be a set of enabled events by processes that are active after E, each about to apply a read, write or CAS primitive. Then there is a schedule $\sigma(E, S)$ of these events such that $\mathcal{M}(E\sigma(E, S)) \leq 3 \cdot \mathcal{M}(E)$.*

PROOF. First, we schedule all *read*, trivial *CAS* and trivial *write* events in an arbitrary order. Let σ_1 denote the resulting execution fragment. No event in σ_1 is visible, hence $\forall o \in B : F(o, E\sigma_1) = F(o, E) \leq \mathcal{M}(E)$. Moreover, for every $p \in P$ that takes a step in σ_1: $AW(p, E\sigma) \leq AW(p, E) + \max_o(\{|F(o, E)| | o \in B\}) \leq 2 \cdot \mathcal{M}(E)$.

We next schedule all remaining *write* events (in an arbitrary order) and let σ_2 denote the resulting execution fragment. Consider all the events of σ_2 that access a specific base object o, if there are any. Only the last of them, denoted by e_l, is visible in $E\sigma_1\sigma_2$; let p_l be the process that issues e_l. Consequently, $F(o, E\sigma_1\sigma_2) = F(o, E) + AW(p_l, E) \leq 2 \cdot \mathcal{M}(E)$. Moreover, for every $p \in P$ that takes a step in σ_2: $AW(p, E\sigma) = AW(p, E) \leq \mathcal{M}(E)$.

Finally, we schedule all remaining *CAS* events (in an arbitrary order) and denote the resulting execution fragment by σ_3. Consider all events of σ_3 that access a specific base object o, if there are any. Let e_f be the first of these events and let p_f be a process that issues e_f. We consider two cases.

1. If o was written in σ_2, then its value after $E\sigma_1\sigma_2$ differs from the *expected* parameter of e_f and of all the other *CAS* events of σ_3. Thus, all these events are trivial, hence: $F(o, E\sigma_1\sigma_2\sigma_3) = F(o, E\sigma_1\sigma_2) \leq 2 \cdot \mathcal{M}(E)$. Moreover, for every $p \in P$ that takes a step in σ_3, $AW(p, E\sigma) \leq AW(p, E) + F(o, E\sigma_1\sigma_2) \leq 3 \cdot \mathcal{M}(E)$.

2. If o was not written in σ_2, then e_f is non-trivial and does changes the value of o, making all consequent *CAS* events in σ_3 trivial. Hence $F(o, E\sigma_1\sigma_2\sigma_3) = F(o, E\sigma_1) + AW(p_f, E) \leq 2 \cdot \mathcal{M}(E)$. Moreover, for every $p \in P$ that takes a step in σ_3: $AW(p, E\sigma) \leq AW(p, E) + F(o, E) + AW(p_f, E) \leq 3 \cdot \mathcal{M}(E)$. ☐

LEMMA 2. *Let E be an execution and let $p \in P$ be a process that issues events in E. Let E' be an execution obtained from E in the following manner. First, all the events issued by p are removed from E. Then, for every process $q \neq p$, all the events of q that are aware of p are removed. Then E' is an execution.*

PROOF. Let $e_q \in E'$ be an event by process q such that $E' = E'_1e_qE'_2$. Since $e_q \in E$ we can write $E = E_1e_qE_2$. If $|E'_1| = k$ we denote E_1 by σ_k and E'_1 by σ'_k. By induction on $k = 1, \cdots |E'|$ we prove that σ'_k is an execution.
Base: σ'_0 is empty and is obviously an execution.
Hypothesis: σ'_k is an execution.
Step: $\sigma'_{k+1} = \sigma'_ke_q$. From induction hypothesis, σ_k is an execution. Assume towards a contradiction that σ_ke_q is not an execution, i.e. e_q returns different responses when scheduled after σ_k and after σ'_k. Then e_q is aware of some event e_1 that was removed from σ_k. However, from Definition 2 and from the way E' is constructed from E, it follows that there is a finite sequence of events $e_q, e_1, \ldots, e_j$, each of which is aware of its successor, such that e_j is an event of p. It follows from Definition 2 that e_q is aware of p and cannot appear in σ'_{k+1}. This is a contradiction. ☐

LEMMA 3. *Let I be a linearizable obstruction-free implementation of a counter and let EE_1 be an execution of I such that each of the processes of $P' = \{p_1, \ldots, p_{N-1}\}$ completes a* `CounterIncrement` *operation in E, and E_1 is an extension of E, in which p_N performs to completion a single* `CounterRead` *operation. Then $|AW(p_N, EE_1)| = N$.*

PROOF. Assume towards a contradiction that $|AW(p_N, EE_1)| < N$. Hence, p_N is not aware of some process $p_i \in P'$. We construct from EE_1 a new execution E' in the following manner. First, we remove from EE_1 all the events of p_i. Then, for each $p_k \in P', p_k \neq p_i$, if any of p_k's events in E is aware of p_i, then we remove all the events of p_k starting from the first event of p_k that is aware of p_i. Since p_N is unaware of p_i, all its events are preserved in E'. From Lemma 2, E' is an execution.

From Definitions 2-3, since p_N is unaware of p_i in EE_1, EE_1 and E' are indistinguishable to p_N. From linearizability, p_N'th `CounterRead` operation returns $N - 1$ in EE_1 but must return a smaller value in E', since at most $N-2$ `CounterIncrement` operations were completed in E'. This is a contradiction. □

PROOF OF THEOREM 1. We iteratively construct an execution $E = \sigma_1\sigma_2, \ldots, \sigma_r$, in which each of the processes of $P' = \{p_1, \ldots, p_{N-1}\}$ completes a `CounterIncrement` operation. For $j \in \{1, \ldots, r\}$, we let $E_j = \sigma_1\sigma_2, \ldots, \sigma_j$. Our construction maintains the invariant that for all $j \in \{1, \ldots, r\}$ and $o \in \mathbf{B}$, $|F(o, E_j)| \leq 3^j$.

In the initial configuration, the familiarity set of all base objects is empty, the awareness set of each process contains only itself, and each of the processes of P' has an enabled event. Let S denote the set of these events. From Lemma 1, there is a schedule $\sigma_1 = \sigma(\perp, S)$ such that $\mathcal{M}(\sigma_1) = \mathcal{M}(E_1) \leq 3$.

Assume we have constructed execution E_i. If a subset $Q \subseteq P'$ of processes did not complete their `CounterIncrement` operations in E_i, then we construct execution $E_{i+1} = E_i\sigma_{i+1}$ as follows. Let S denote the set of the events of the processes of Q that are enabled after E_i. From Lemma 1, there is a schedule $\sigma_{i+1} = \sigma(E_i, S)$ such that $\mathcal{M}(E_{i+1}) \leq 3\mathcal{M}(E_i) \leq 3^{i+1}$. We proceed in this manner until all the processes of P' complete their operations or until we complete $\lceil \log_3 \frac{N}{f(N)} \rceil$ iterations, whatever occurs first.

Let p_l be a process that does not terminate its `CounterIncrement` operation in E_{r-1}. There must be such a process, as otherwise the construction of E would have stopped after the $(r-1)$'th iteration. Clearly from our construction, p_l issues r events in E. To prove the theorem, we will show that $r = \Omega\left(\log_3 \frac{N}{f(N)}\right)$.

Assume towards a contradiction that $r = o\left(\log_3 \frac{N}{f(N)}\right)$, implying that all the processes of P' completed their `CounterIncrement` operations in E. From our construction, the following holds:

$$\forall o \in \mathbf{B} : |F(o, E)| \leq 3^r = o\big(\frac{N}{f(N)}\big). \qquad (1)$$

Let E_1 be an extension of E in which p_N performs a `CounterRead` operation. Since p_N performs $O(f(N))$ steps in E_1, it accesses at most $O(f(N))$ base objects. Therefore, it follows from Equation 1 that $|AW(p_N, EE_1)| = o(N)$. This is a contradiction to Lemma 3. □

Given an N-component single-writer snapshot object, it is straightforward to implement a counter as follows. To perform a `CounterIncrement` operation, process p_i increments the value of the i'th component by performing a single `Update` operation. To read the counter, a process performs a single `Scan` operation and returns the sum of all components. We get the following:

COROLLARY 1. *Let I be an N-process obstruction-free implementation of a single-writer snapshot object from the* read*,* write *and* CAS *primitives. If the step-complexity of* `Scan` *is $O(f(N))$, then the step-complexity of* `Update` *is $\Omega\left(\log \frac{N}{f(N)}\right)$.*

Jayanti presented an *f-arrays*-based [14] construction of counters and snapshots where `CounterRead` and `Scan` operations have constant step complexity, and `CounterIncrement` and `Update` operations have logarithmic step complexity. His construction uses the *load-link/store-conditional* primitives in addition to reads and writes, but can be made to work also using CAS instead.

It follows from Theorem 1 and Corollary 1 that no constant-read implementation for these objects from read, write and CAS can have update operations with sub-logarithmic step complexity. This follows also from another work of Jayanti [13] where he shows that the sum of steps performed by a `CounterIncrement` operation followed by a `CounterRead` operation (or an `Update` operation followed by a `Scan` operation) is logarithmic. The following theorem, however, does not follow from [13].

THEOREM 2. *Let I be a obstruction-free N-process implementation of a counter (respectively single-writer snapshot) object from reads and writes that supports only a limited (polynomial in N) number of* `CounterIncrement` *(respectively* `Update`*) operations. If the worst-case step complexity of I's* `CounterRead` *(respectively* `Scan`*) operations is optimal (that is, $O(\log N)$), then the worst-case step-complexity of its* `CounterIncrement` *(respectively* `Update`*) operations is $\Omega(\log N)$.*

PROOF. Immediate from Theorem 1 and Corollary 1. □

4. TRADEOFF FOR MAX REGISTERS

In this section, we prove the following tradeoff.

THEOREM 3. *Let I be an N-process obstruction-free implementation of an M-bounded* maxRegister *from the* read*,* write *and* CAS *primitives. Let $K = min(M, N)$. If the step complexity of* `ReadMax` *is $O(f(K))$, then there is an execution of I in which each of $\Omega\big(f(K)\big)$ processes takes $\Omega(\log \frac{\log K}{\log f(K)})$ steps as it performs a single* `WriteMax` *operation.*

Our proofs combine and extend techniques from [1, 6, 15]. We start by describing our proof strategy. This is followed by formal proofs.

We construct an execution involving $\Theta\big(f(K)\big)$ processes, each performing $\Omega\big(\log \frac{\log K}{\log f(K)}\big)$ steps. The construction proceeds iteratively, where iteration i constructs execution E_i. Initially, we have a set $P' = \{p_1, \cdots, p_{K-1}\}$ of $K-1$ processes such that, for $i \in \{1, \ldots, K-1\}$, p_i is about to perform a `WriteMax` (i) operation.

A key concept in our construction is that of an *essential set*. Each execution E_i is associated with an essential set $\mathcal{E}_i \subseteq P'$ (the *essential set of* E_i) of size $\Omega(\sqrt[2^i]{K})$ that satisfies the following properties: 1) Each process in $\mathcal{E}_i$ performs exactly i steps in E_i. 2) If p is in $\mathcal{E}_i$, then no $q \neq p$ is aware of p after E_i. 3) No base object is familiar with more than a single process in $\mathcal{E}_i$ after E_i. 4) The identifiers of the processes of $\mathcal{E}_i$ are larger than those of all other processes that issue events in E_i.

The properties stated above guarantee that not too many of the processes of $\mathcal{E}_i$ may complete their operations in E_i. To see why, consider a set $F \subseteq \mathcal{E}_i$ of m such processes. As we prove, a `ReadMax` operation Φ by p_K, scheduled after E_i, must access m distinct base objects in the course of its execution - those base objects that are familiar with the processes of F. If Φ fails to read even one such base object (say, the one familiar with p_j), then we can remove all the events issued by $\mathcal{E}_i \backslash \{p_j\}$ from $E_i\Phi$. In this case, Φ will fail to return j, which is the largest value written prior to Φ's execution. It follows that at most $O(f(n))$ processes of $\mathcal{E}_i$ can terminate in E_i.

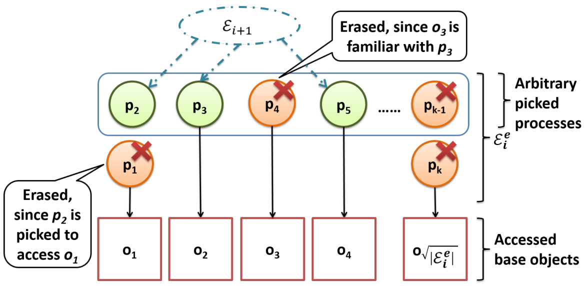
Let us denote by $\mathcal{E}_i^e$ the set of those processes of $\mathcal{E}_i$ that remain active after E_i. Our goal is to pick a relatively large subset $\mathcal{E}_{i+1} \subseteq \mathcal{E}_i^e$, each of whose processes can perform an additional step, while limiting information flow so that properties 1)-4) above hold for $\mathcal{E}_{i+1}$. We now describe how we select $\mathcal{E}_{i+1}$ and construct E_{i+1}.

First, all the events of the processes of $\mathcal{E}_i \setminus \mathcal{E}_i^e$ are removed. We say that these processes are *erased from the execution* (or simply *erased*). Then, we consider the next event that will be issued by the processes of $\mathcal{E}_i^e$. Specifically, we consider the number j of distinct base-objects that these events will access. The following two cases exist.

Case 1 (Low Contention): If $j > \sqrt{|\mathcal{E}_i^e|}$, we pick an arbitrary set A' of $\sqrt{|\mathcal{E}_i^e|}$ processes accessing *distinct* base objects. We then pick a subset $A \subseteq A'$ such that no step by a process of A is about to access a base object that has another process of A in its familiarity set. As we show, a constant fraction of the processes of A' remain in A. The set A is defined as $\mathcal{E}_{i+1}$.

To construct E_{i+1}, the processes of $\mathcal{E}_i^e \setminus A$ are also erased. E_{i+1} is obtained by extending the remaining execution by allowing each of the processes of $\mathcal{E}_{i+1}$ to perform an additional step. We refer to this case as the *low-contention scenario*. Figure 1 illustrates the construction of $\mathcal{E}_{i+1}$ in the low-contention scenario.

Figure 1: Low contention case for iteration i

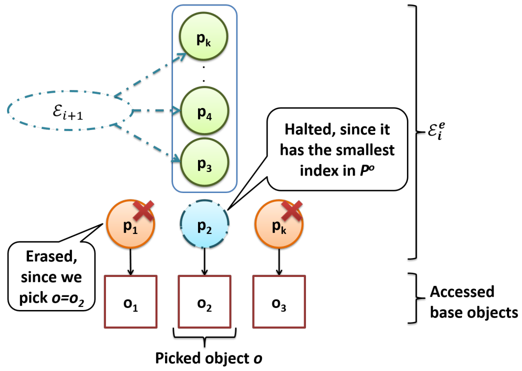


The processes that are not *erased* from the execution form the *essential set* $\mathcal{E}_{i+1}$.

Case 2 (High Contention): Otherwise $\left(j \leq \sqrt{|\mathcal{E}^e|}\right)$, we pick an arbitrary base object o accessed by a subset (denoted P^o) of at least $\sqrt{|\mathcal{E}^e|}$ processes from $\mathcal{E}_i^e$. We erase all the processes of $\mathcal{E}_i \backslash P^o$ from the execution. As for the processes of P^o, there are two possibilities. Either we can schedule their steps so that none of them becomes visible on o (in which case they will all belong to $\mathcal{E}_{i+1}$); or we schedule them so that a *single* process, say p_k, becomes visible and a constant fraction of the processes of P^o have bigger identifiers. These are the processes that will form $\mathcal{E}_{i+1}$. As for p_k, we say that it is *halted*. It will not issue additional events in later iterations, nor will it be a part of essential sets of later iterations. The rest of the processes of P^o are erased from the execution. We refer to this case as the *high contention* scenario. Figure 2 illustrates the construction of $\mathcal{E}_{i+1}$ in the high-contention scenario.

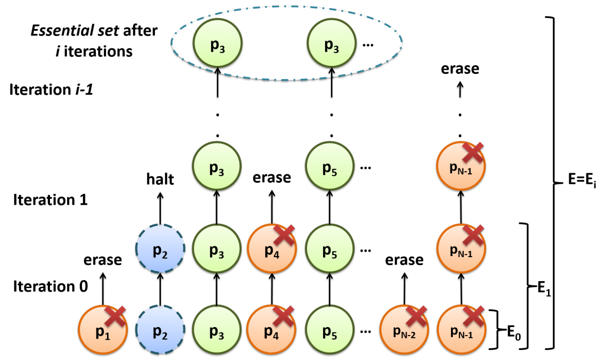
The high-level structure of our construction is depicted in Figure 3. As we show, the construction may proceed for $\Omega(\log \frac{\log K}{\log f(K)})$ iterations. Before providing the formal proofs, we give a few required definitions. In the following, we let E

Figure 2: High contention case for iteration i



The processes not in P^o are *erased* from the execution. Process p_2 becomes *halted*. The rest of the processes in P^o form the *essential set* $\mathcal{E}_{i+1}$.

Figure 3: The construction of execution E



Erased processes do not appear in E. *Halted* processes issue an event in each construction iteration until they become halted. *Essential set* processes issue an event in every construction iteration.

denote an execution of a max register implementation I and we let P be the set of processes taking steps in E.

Definition 5. *Let O be the set of all base objects used by I. We say that a process $p \in P$ is* hidden *after E, if $\forall p' \in P : p \in AW(p', E) \rightarrow p = p'$ (informally, no process except p is aware of p after E). We say that $P' \subseteq P$ is a* hidden set *after E, if the processes of P' are* hidden *after E and if $\forall o \in O : |F(o, E) \cap P| \leq 1$ (informally, each base object in O is familiar with at most a single process in P after E).*

Claim 1. *Let P' be a set of processes hidden after E, then $E' = E^{-P'}$ is an execution.*

Proof. From Definition 5, for all $p \in P'$, no process but p is aware of p after E, hence the claim follows from Lemma 2. ☐

Definition 6. *We say that $P' \subseteq P$ is a* supreme set*, if $\forall p_i \in P', \forall p_j \in P \backslash P' : i > j$ holds (informally, the processes of P' have the highest indices out of all processes that issue events in the execution).*

DEFINITION 7. *We say that $P' \subseteq P$ is an i-step* essential set *of E, if P' is a supreme set hidden after E, such that every process in it issues exactly i events in E.*

LEMMA 4. *Let E_i be an execution and let $\mathcal{E}_i$ be an i-step essential set of E_i. Let $\mathcal{E}^e \subseteq \mathcal{E}_i$ be the set of those processes of $\mathcal{E}_i$ that have an enabled event after E_i and let $m = |\mathcal{E}^e|$ denote its size. If $m \geq 81$, then there exists an execution E_{i+1} with an $(i+1)$-step essential set of size at least $\frac{\sqrt{m}}{3} - 2$.*

PROOF. Let $S = \{e_1 \ldots e_m\}$ be the set of events that the processes of $\mathcal{E}^e$ are about to issue after E_i. Let O denote the set of all base objects accessed by the events of S. Also, for $o \in O$, let $P^o \subseteq \mathcal{E}^e$ denote the set of all processes from $\mathcal{E}^e$ that are about to access o. There are two cases to consider.

Case 1 (Low Contention): For each base object $o \in O$: $|P^o| \leq \sqrt{m}$. In this case the steps by $\mathcal{E}^e$ access $k \geq \sqrt{m}$ distinct base objects. For every accessed object o we arbitrary pick a process $p^o \in P^o$ and denote the resulting set by $P"$. We build an undirected graph $G =< V, E >$ such that $V = \{v^o | p^o \in P"\}$ and $E = \{< v^o, v^{o'} > | p^{o'} \in F(o, E_i)\}$. It follows immediately from the construction that $|V| = |P"| = k$.

$P" \subseteq \mathcal{E}_i$, hence from Definitions 5 and 7, $\forall o \in O$: $|F(o, E_i) \cap P"| \leq 1$, i.e $|E| \leq k$. Consequently, the average degree of G is $d \leq 2$. From Turán's theorem [9], there is an independent set $V' \subseteq V$ such that $|V'| = \lceil \frac{k}{3} \rceil$. Let $\mathcal{E}_{i+1} = \{p^o | v^o \in V'\}$ and let σ be a sequence of events in which we schedule all the events by the processes of $\mathcal{E}_{i+1}$ (in some arbitrary order). Let $K = \mathcal{E}_i \backslash \mathcal{E}_{i+1}$. From Claim 1, $E_{i+1} = E_i{}^{-K}\sigma$ is an execution. We now show that $\mathcal{E}_{i+1}$ is an $(i+1)$-step essential set of E_{i+1}.

No process $p^o \in \mathcal{E}_{i+1}$ becomes aware of $p^{o'} \in \mathcal{E}_{i+1}$ in σ, otherwise $p^{o'} \in F(o, E_i)$, i.e $< v^o, v^{o'} > \in E$ in contradiction to the fact that V' is an independent set. Moreover, every process in σ accesses a distinct object, i.e. $F(o, E_i\sigma) \subseteq F(o, E_i) \cup AW(p^o, E_i)$. If $p^{o'} \in \mathcal{E}_{i+1} \cap F(o, E_i\sigma)$, then $p^{o'} \in F(o, E_i)$, i.e $< v^o, v^{o'} > \in E$ in contradiction to the fact that V' is an independent set. Consequently, o is familiar with at most a single process of $\mathcal{E}_{i+1}$ after E_{i+1}. It follows from Definition 5 that $\mathcal{E}_{i+1}$ is hidden after E_{i+1}.

$\mathcal{E}_i$ is an essential set of E_i, $\mathcal{E}_{i+1} \subset \mathcal{E}_i$ and all the processes of $\mathcal{E}_i \backslash \mathcal{E}_{i+1}$ are erased, hence $\mathcal{E}_{i+1}$ is a supreme set of E_{i+1}. Since $\mathcal{E}_i$ is an i-step essential set of E_i, every process of $\mathcal{E}_i$ issues exactly i events in E_i. It follows immediately from our construction of E_{i+1} that every process of $\mathcal{E}_{i+1}$ issues exactly $i+1$ events in E_{i+1}. Thus, $\mathcal{E}_{i+1}$ is an $(i+1)$-step essential set. Finally, it is immediate from our construction that $|\mathcal{E}_{i+1}| \geq \frac{\sqrt{m}}{3} - 1$.

Case 2 (High Contention): There is a base object $o \in O$ such that $|P^o| \geq \sqrt{m} + 1$. We consider the processes of P^o according to the type of operation they are about to apply to o after E_i: we let ${P^o}_C \subseteq P^o$ denote those processes about to apply a CAS event that will change the value of o (if applied immediately after E_i); we let ${P^o}_W \subseteq P^o$ denote those processes about to apply a *write* event, and we let ${P^o}_T \subseteq P^o$ denote those processes about to apply a *read* or a CAS event that will not change the value of o. We need to consider the following three sub-cases.

Sub-case 1: $|{P^o}_C| \geq \frac{\sqrt{m}}{3}$. Let $S = F(o, E_i) \cap \mathcal{E}^e$. After execution E_i, object o is familiar with at most a single process of $\mathcal{E}^e$, thus $|S| \leq 1$. Let p_l be the process with the smallest identifier of all the processes of ${P^o}_C$ and let $\mathcal{E}_{i+1} = {P^o}_C \backslash (\{p_l\} \cup S\})$ (since $m \geq 81$, $\mathcal{E}_{i+1}$ is non-empty). Let σ be a sequence of events in which we schedule all the *CAS* events by the processes of $\mathcal{E}_{i+1}$ (in some arbitrary order), preceded by the non-trivial *CAS* by p_l. Let $K = (\mathcal{E}_i \backslash {P^o}_C) \cup S$. From Claim 4, $E_{i+1} = E_i{}^{-K}\sigma$ is an execution. We now show that $\mathcal{E}_{i+1}$ is an $(i+1)$-step essential set of E_{i+1}.

Since the *CAS* event by p_l changes the value of o, the *CAS* events by the processes of $\mathcal{E}_{i+1}$ are trivial and invisible on o after σ. Hence, all the processes of $\mathcal{E}_{i+1}$ become aware in σ only of processes of $F = (F(o_i, E_i) \backslash K) \cup \{p_l\}$. Since $F \cap \mathcal{E}_{i+1} = \emptyset$, these processes do not become aware of one another in σ. Moreover, no base object except o becomes familiar with a new process, and o's familiarity set may only be extended by $AW(p_l, E_i)$, hence o is familiar with no process of $\mathcal{E}_{i+1}$ after E_{i+1}. It follows from Definition 5 that $\mathcal{E}_{i+1}$ is hidden after E_{i+1}.

$\mathcal{E}_i$ is an essential set of E_i, $\mathcal{E}_{i+1} \subset \mathcal{E}_i$, all the processes of $\mathcal{E}_i \backslash \mathcal{E}_{i+1}$ except for p_l are erased and p_l's identifier is smaller than the identifiers of the processes of $\mathcal{E}_{i+1}$, hence $\mathcal{E}_{i+1}$ is a supreme set of E_{i+1}. Since $\mathcal{E}_i$ is an i-step essential set of E_i, every process of $\mathcal{E}_i$ issues exactly i events in E_i. It follows immediately from our construction of E_{i+1} that every process of $\mathcal{E}_{i+1}$ issues exactly $i+1$ events in E_{i+1}. Thus, $\mathcal{E}_{i+1}$ is an $(i+1)$-step essential set. Finally, it is immediate from our construction that $|\mathcal{E}_{i+1}| \geq \frac{\sqrt{m}}{3} - 2$.

Sub-case 2: $|{P^o}_W| \geq \frac{\sqrt{m}}{3}$. Let p_l be the process with the smallest identifier of all the processes of ${P^o}_W$ and let $\mathcal{E}_{i+1} = {P^o}_W \backslash \{p_l\}$ (since $m \geq 81$, $\mathcal{E}_{i+1}$ is non-empty). Let σ be a sequence of events in which we schedule all *write* events (in some arbitrary order) by the processes of $\mathcal{E}_{i+1}$ followed by the *write* by p_l. Let $K = \mathcal{E}_i \backslash {P^o}_W$. Since $\mathcal{E}_i$ is hidden after E_i, it follows from Claim 4 that $E_{i+1} = E_i{}^{-K}\sigma$ is an execution. We now show that $\mathcal{E}_{i+1}$ is an $(i+1)$-step essential set of E_{i+1}.

Since σ consists of write events, no process becomes aware of another process in σ. Moreover, no base object except o becomes familiar with a new process, and o's familiarity set may only be extended by $p_l \notin \mathcal{E}_{i+1}$, hence o is familiar with at most a single process of $\mathcal{E}_{i+1}$ after E_{i+1}. It follows from Definition 5 that $\mathcal{E}_{i+1}$ is hidden after E_{i+1}.

$\mathcal{E}_i$ is an essential set of E_i, $\mathcal{E}_{i+1} \subset \mathcal{E}_i$ and p_l's identifier is smaller than the identifiers of the processes of $\mathcal{E}_{i+1}$, hence $\mathcal{E}_{i+1}$ is a supreme set of E_{i+1}. Since $\mathcal{E}_i$ is an i-step essential set of E_i, every process of $\mathcal{E}_i$ issues exactly i events in E_i. It follows immediately from our construction of E_{i+1} that every process of $\mathcal{E}_{i+1}$ issues exactly $i+1$ events in E_{i+1}. Thus, $\mathcal{E}_{i+1}$ is an $(i+1)$-step essential set.

Sub-case 3: $|{P^o}_T| \geq \frac{\sqrt{m}}{3}$**.** Let $S = F(o, E_i) \cap \mathcal{E}^e$. After execution E_i, object o is familiar with at most a single process of $\mathcal{E}^e$, thus $|S| \leq 1$. Let $\mathcal{E}_{i+1} = {P^o}_T \backslash S$ (since $m \geq 81$, $\mathcal{E}_{i+1}$ is non-empty) and let σ be a sequence of events in which we schedule all *read* and trivial *CAS* events by the processes of $\mathcal{E}_{i+1}$ in some arbitrary order. Let $K = (\mathcal{E}_i \backslash {P^o}_T) \cup S$. Since $\mathcal{E}_i$ is hidden after E_i, it follows from Claim 4 that $E_{i+1} = E_i{}^{-K}\sigma$ is an execution. We now show that $\mathcal{E}_{i+1}$ is an $(i+1)$-step essential set of E_{i+1}.

All the processes of $\mathcal{E}_{i+1}$ become aware in σ only of processes in $F = F(o_i, E_i) \backslash K$. Since $F \cap \mathcal{E}_{i+1} = \emptyset$, these processes do not become aware of one another in σ. Moreover, all events in σ are trivial, thus the familiarity set of o (or any other object) is not changed. It follows from Definition 5 that $\mathcal{E}_{i+1}$ is hidden after E_{i+1}.

$\mathcal{E}_i$ is an essential set of E_i, $\mathcal{E}_{i+1} \subset \mathcal{E}_i$ and all the processes of $\mathcal{E}_i \backslash \mathcal{E}_{i+1}$ are erased, hence $\mathcal{E}_{i+1}$ is a supreme set of E_{i+1}. Since $\mathcal{E}_i$ is an i-step essential set of E_i, every process of $\mathcal{E}_i$ issues exactly i events in E_i. It follows immediately from our construction of E_{i+1} that every process of $\mathcal{E}_{i+1}$ issues exactly $i+1$ events in E_{i+1}. Thus, $\mathcal{E}_{i+1}$ is an $(i+1)$-step essential set. Finally, it is immediate from our construction that $|\mathcal{E}_{i+1}| \geq \frac{\sqrt{m}}{3} - 1$. $\Box$

LEMMA 5. *Let E be an execution in which a process p_i that is hidden after E completes its* `WriteMax` *operation. Assume also that p_i wrote a (unique) maximum value in E. Let p_j be a process that issued no events in E and let Φ be an execution of a* `ReadMax` *operation by p_j immediately after E. Then p_j must access in Φ an object familiar with p_i.*

PROOF. Assume towards a contradiction that p_j accesses no such object. Consider execution E' obtained from E by removing all the events by p_i from E. From Claim 1, E' is an execution. Since p_i is hidden in E, E' is indistinguishable from E for p_j. Consequently, p_j performs Φ also after E' and returns the same response in both $E\Phi$ and $E'\Phi$. This is a contradiction since the maximum values written in E and E' differ. $\Box$

LEMMA 6. *Let E be an execution in which each process $p_i \in P \subseteq \{p_1, \cdots, p_{K-1}\}$ performs a* `WriteMax`*(i) operation. Let A be a a hidden and supreme set of E and let $\mathcal{E}^c \subseteq A$ denote the set of those processes of A that complete their operation in E. If the step complexity of* `ReadMax` *operations is at most m, then $|\mathcal{E}^c| \leq m$.*

PROOF. We iteratively construct an execution E' such that, if $|\mathcal{E}^c| > m$, then process p_K must miss the maximum value written to the max object when it performs its `ReadMax` operation after E'. We denote the execution obtained after r iterations as E'_r. Our construction starts with $E'_0 = E^{-A\backslash\mathcal{E}^c}$. From Claim 1, E'_0 is an execution.

We construct E'_r from E'_{r-1} as follows. Let e_r be the r-th event about to be issued by p_K after E'_{r-1} and let o be the object it will access. We let $I = F(o, E'_{r-1}) \cap \mathcal{E}^c$ and $E'_r = {E'_{r-1}}^{-I} e_r$. Since $\mathcal{E}^c$ is a hidden set, $|I| \leq 1$, hence E'_r is either $E'_{r-1}e_r$ or ${E'_{r-1}}^{-\{p_i\}}e_r$ for some $p_i \in \mathcal{E}^c$. Since E'_{r-1} is an execution and since p_i (if it exists) is hidden, it follows from Claim 1 that E'_r is an execution. Since none of the events of p_K access an object that is familiar with a process of A, $E'_r e_1 \cdots e_r$ is an execution as well. The construction stops after p_K takes its last step, hence it stops after at most m iterations.

Assume towards a contradiction that $|\mathcal{E}^c| > m$. Let K be the set of processes erased in the course of the construction of E'. Since $|K| \leq m$, the set $\mathcal{E}^c \backslash K$ is non-empty. Let p_{max} be the process in $\mathcal{E}^c \backslash K$ with maximum ID. Since $p_{max} \in \mathcal{E}^c \subseteq A$ and from construction, p_{max} is the process that performed in E' the `WriteMax` operation with the maximum operand. Moreover, p_{max} completed its operation in E'.

From Lemma 5, p_K must access an object familiar with p_{max}. However, our construction of E' ensures that none of the objects accessed by p_K is familiar with a process of $\mathcal{E}^c$. This is a contradiction. $\Box$

PROOF OF THEOREM 3. We construct an execution E with the required properties. The construction proceeds in iterations. In iteration i, we construct an execution E_i that has an i-step essential set $\mathcal{E}_i$ of size $\Omega(K^{1/3^i})$. Initially, we have a set $P' = \{p_1, \cdots, p_{K-1}\}$ of $K-1$ processes such that, for $i \in \{1, \ldots, K-1\}$, p_i is about to perform a `WriteMax` (i) operation.

For the base case, we let E_0 be the empty execution and claim that P' is a 0-step essential set. In the initial configuration, all processes are aware only of themselves and all base objects have empty familiarity sets. Hence, all processes are hidden after the empty execution . Moreover, P' is supreme and all processes in it issue 0 events in E_0. It follows from Definition 7 that P' is a 0-step essential set of E_0.

The construction stops after the first iteration i^*, such that at least half of the processes of $\mathcal{E}_{i^*}$ terminate in E_{i^*}, or $|\mathcal{E}_{i^*+1}| < f(K)$, whichever happens first. In the first case, from Lemma 6:

$$|\mathcal{E}_{i^*}| \leq 2 \cdot f(K). \tag{2}$$

In the second case, we get from Lemma 4 and from our construction:

$$|\mathcal{E}_{i^*}| = O(|\mathcal{E}_{i^*+1}|)^2 = O(f(K))^2. \tag{3}$$

On the other hand, from Lemma 4, we have:

$$\begin{aligned} \forall i \in \{1, \ldots, i^*\} : |\mathcal{E}_i| \geq \frac{1}{2}(|\mathcal{E}_{i-1}|)^{1/2} - 2 = \Omega(|\mathcal{E}_{i-1}|)^{1/3} \\ \implies |\mathcal{E}_i| = \Omega(K^{1/3^i}). \end{aligned} \tag{4}$$

Combining Equations 2, 3 and the right-hand equality of Equation 4, we get $i^* = \Omega(\log \frac{\log K}{\log f(K)})$. The theorem now follows from the fact that each process in $\mathcal{E}_{i^*}$ issues i^* events in E_{i^*} and the fact that, from our construction, $|\mathcal{E}_{i^*}| \geq f(K)$. $\Box$

The following is immediate from Theorem 3.

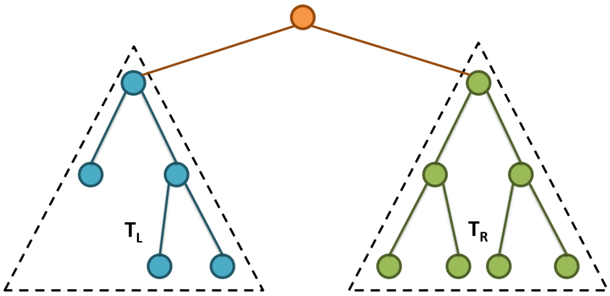
THEOREM 4. *Let I be a obstruction-free N-process implementation of an M-bounded max register from read, write and CAS. If the worst-case step complexity of I's* `ReadMax` *operation is $O(\log M)$, then the worst-case step-complexity of its* `WriteMax` *operation is $\Omega\big(\log\log min(N, M)\big)$.*

5. MAX REGISTER IMPLEMENTATIONS USING READ/WRITE/CAS

We now present a wait-free *maxRegister* algorithm, using *read*, *write* and *CAS* primitives. The step-complexity of `ReadMax` operations is constant and the step complexity of a `WriteMax`(v) operation is $O\big(\log min(N, v)\big)$. The algorithm uses a binary tree T, every node of which stores an integer *value* (initialized to $-\infty$) and pointers to its child nodes and parent node. The left sub-tree of T, denoted as T_L, is an unbalanced binary tree with $N-1$ leaves such that its i-th leaf is at depth $O(\log i)$ (the construction of such a tree, referred to as a B_1 tree, is introduced in [8]). The right sub-tree of T, denoted as T_R, is a complete binary tree with N leaves. An illustration of the data structure for $N = 4$ processes is depicted in Figure 4.

The pseudo-code appears in Algorithm A. To perform a `ReadMax`, a process simply reads the *value* stored in the root. To perform a `WriteMax` (v) operation, process p_i writes v to a leaf $\mathcal{L}$ and attempts to propagate it up to the root of T in a manner we soon explain. $\mathcal{L}$ is selected as follows: if $v < N$

Figure 4: The data structure for *maxRegister* shared by $N = 4$ processes



T_L is an unbalanced B_1 tree with $N - 1$ leaves. T_R is a complete binary tree with N leaves.

Algorithm A Max Register

```
 1: function READ
 2:     return T.root.value
 3: procedure PROPAGATE(n)
 4:     while n.parent ≠ nil do
 5:         n ← node.parent
 6:         for i = 1 to 2 do
 7:             old_value ← n.value
 8:             new_value ← max(n.left_child.value,
                                n.right_child.value)
 9:             CAS(n.value, old_value, new_value)
10: procedure Write_i(value)
11:     if value < N then
12:         ℒ ← T_L.leaves[value]
13:     else
14:         ℒ ← T_R.leaves[i]
15:         old_value ← ℒ.value
16:         if value ≤ old_value then return
17:     ℒ.value ← value
18:     PROPAGATE(ℒ)
```

then $\mathcal{L}$ is the v-th leaf of T_L, otherwise it is the i-th leaf of T_R.

Propagating a value given as operand to a `WriteMax` operation up the tree is implemented by the *Propagate* procedure, which works similarly to the *Tree Algorithm* of Jayanti [14]. At each level of the tree along the path from the leaf to the root, a process computes the maximum of the value of its current node and its sibling and attempts to write this maximum to the parent node by using *CAS*. Since the *CAS* may fail, the computation of the maximum value and the *CAS* are performed twice at each level. This ensures that if the *CAS* failed, then a *CAS* by another process must have succeeded in updating the parent node based on the new value.

Now we prove that Algorithm A is correct and wait-free, that the step-complexity of the `ReadMax` operation is constant and that the step complexity of the `WriteMax`(v) operation is $O(\min(\log N, \log v))$.

Linearizability.

In the following, we prove that Algorithm A is *linearizable* [12]. For every operation Φ_i in an execution E of A, we uniquely identify a *linearization point*, i.e. an event $e_i \in E$ within the time interval of Φ_i. For simplicity and WLOG, we assume that no process tries to write a value which is smaller than the values it has written earlier.

DEFINITION 8. *We say that a node $\mathcal{N}$* counts *an operation Φ after E, if $E = E'eE"$, e is a* visible *event on $\mathcal{N}.value$ and either e is issued in Φ or it was issued by some process p such that*

1. *$E' = E_1 e' E_2$, where e' is a* read *by p of $\mathcal{N}'.value$.*
2. *$\mathcal{N}'$ is a child of $\mathcal{N}$ and $\mathcal{N}'$* counts *Φ after E_1.*

We denote the set of all operations counted by $\mathcal{N}$ after E as $\mathcal{C}(N, E)$.

Clearly, if $\mathcal{N}$ counts Φ after E, then it counts Φ in all extensions of E; and if $\mathcal{N}$ does not count Φ after E, then it does not count Φ in any prefix of E.

DEFINITION 9. *For `ReadMax` operation Φ_r in E we define the* linearization point *as the* read *event on $T.root.value$ (line 2). For `WriteMax` operation Φ_w in E we define the* linearization point *as the first event e after which $T.root$ counts Φ_w, i.e. if $E = E_1 e E_2$ then $\Phi_w \notin \mathcal{C}(T.root, E_1)$ and $\Phi_w \in \mathcal{C}(T.root, E_1 e)$.*

Note that a `WriteMax` operation may be linearized by an event of another `WriteMax` operation.

We let H be the sequential history obtained from E according to the linearization points specified above. We let $e_1 \overset{E}{\prec} e_2$ denote that event e_1 *precedes* event e_2 in E. Moreover, we say that event e *precedes* (is *preceded* by) operation Φ in E if it is issued before the first (after the last) event of Φ and denote this by $e \overset{E}{\prec} \Phi$ ($\Phi \overset{E}{\prec} e$). We now prove the following:

1. H preserves the partial order of operations in E. It is sufficient to prove that the *linearization point* e of every operation Φ is defined uniquely within the time interval of Φ, i.e $\neg(e \overset{E}{\prec} \Phi)$ and $\neg(\Phi \overset{E}{\prec} e)$.
2. H is legal sequential history of *maxRegister*. It is sufficient to prove that every `ReadMax` operation in H returns the maximum value written before it in H.

OBSERVATION 1. *By Definition* 9*, the* linearization point *e of `ReadMax` operation Φ is a* read *issued by Φ on $T.root.value$. Obviously, e is within the time interval of Φ and is defined uniquely.*

OBSERVATION 2. *By Definition* 9*, the* linearization point *e of `WriteMax` operation Φ is the first and thus unique event, after which Φ is counted by $T.root$.*

LEMMA 7. *Let Φ be a `WriteMax` operation in E such that its* linearization point *e, as specified by Definition* 9*, is also in E. Then e does not precede Φ in E.*

PROOF. Assume towards a contradiction that $e \overset{E}{\prec} \Phi$, i.e. $\Phi \in \mathcal{C}(T.root, E'e)$, where $E'e$ is a prefix of E. Since no events of Φ are issued in $E'e$, from Definition 8, some child of $T.root$ counts Φ in $E'e$. We apply this argument recursively until it implies that some leaf $\mathcal{L}$ should count Φ in $E'e$. Consequently, from Definition 8, $\mathcal{L}$ must be written by Φ in $E'e$. On the other hand, Φ issues no events in $E'e$. This is a contradiction. $\square$

Let $\mathcal{N}$ be some internal node of T and let p_j be a process that accesses $\mathcal{N}$ when executing a `WriteMax` operation Φ. For $i \in \{1,2\}$ we denote the i-th *read* of $\mathcal{N}.value$ by p_j in line 7 as $read_j^i(\mathcal{N},\Phi)$, the i-th *read* of $\mathcal{N}.left\_child.value$ by p_j in line 8 as $read\_lc_j^i(\mathcal{N},\Phi)$, the i-th *read* of $\mathcal{N}.right\_child.value$ by p_j in line 8 as $read\_rc_j^i(\mathcal{N},\Phi)$ and the i-th *CAS* of $\mathcal{N}.value$ by p_j in line 9 as $cas_j^i(\mathcal{N},\Phi)$. The following lemma will help us prove that a `WriteMax` operation does not precede its *linearization point.*

LEMMA 8. *In execution E, the sequence of values stored in every node of T is non-decreasing.*

PROOF. First, we consider values stored in the leaves of T. The v-th leaf of T_L stores either $-\infty$ (initial vale) or $v \geq 0$ (when written in line 17), hence its values are non-decreasing. The i-th leaf of T_R is written only by p_i, hence it only stores the operands of `WriteMax` operations by p_i, that are always non-decreasing.

We now consider the internal nodes of T. Assume towards a contradiction that the claim of the lemma is violated in E and let event e, issued by some process p_i, be the first event violating the claim, by changing $\mathcal{N}.value$, for some node $\mathcal{N}$, from value x to a smaller value y. Obviously e is a non-trivial *CAS* event issued by p_i when executing some `WriteMax` operation Φ_i. WLOG, e is a $cas_i^1(\mathcal{N},\Phi_i)$ and y is the maximum of values obtained by $read\_lc_i^1(\mathcal{N},\Phi_i)$ and $read\_rc_i^1(\mathcal{N},\Phi_i)$.

Since e changes the value of $\mathcal{N}.value$, according to the algorithm A, x is the value obtained by $read_i^1(\mathcal{N},\Phi_i)$. Let p_j be the process that wrote x to $\mathcal{N}.value$, when executing a `WriteMax` operation Φ_j. WLOG, p_j obtained x from $read\_lc_j^1(\mathcal{N},\Phi_j)$ and wrote it to $N.value$ in $cas_j^1(\mathcal{N},\Phi_j)$. It holds that,

$$read\_lc_j^1(\mathcal{N},\Phi_j) \overset{E}{\prec} cas_j^1(\mathcal{N},\Phi_j) \overset{E}{\prec} read_i^1(\mathcal{N},\Phi_i) \overset{E}{\prec} read\_lc_i^1(\mathcal{N},\Phi_i) \overset{E}{\prec} read\_rc_i^1(\mathcal{N},\Phi_i).$$

On the other hand, $read\_lc_j^1(\mathcal{N},\Phi_j)$ returns x, while $read\_lc_i^1(\mathcal{N},\Phi_i)$ returns $y' \leq y < x$. Hence the maximal value stored in $\mathcal{N}$'s left child has decreased in the execution prior to E. Thus e is not the first event violating the claim. This is a contradiction. □

LEMMA 9. *Let Φ be a `WriteMax` operation in E such that its* linearization point *e, as specified by Definition* 9, *is also in E. Then Φ does not precede e in E.*

PROOF. If Φ does not complete in E then obviously $\neg(e \overset{E}{\prec} \Phi)$. Hence we only consider operations that complete in E. Let E' be a prefix of E such that Φ completes in E'. We prove that $\Phi \in \mathcal{C}(T.root, E')$. We consider this claim as a special case of the following invariant.

INVARIANT 1. *Let $e^{\mathcal{N}}$ be the last* write *or* CAS *event of Φ that accesses some node $\mathcal{N}$ in E. Let $E = E_1^{\mathcal{N}} e^{\mathcal{N}} E_2^{\mathcal{N}}$. Then $\Phi \in \mathcal{C}(\mathcal{N}, E_1^{\mathcal{N}} e^{\mathcal{N}})$*

Let $\mathcal{N}_1, \cdots, \mathcal{N}_d$, where d is the depth of the leaf $\mathcal{L}$ written by Φ, denote the nodes in the propagation path of Φ, such that $\mathcal{N}_1$ is $\mathcal{L}$ and $\mathcal{N}_d$ is $T.root$. We prove by induction on $r = 1, \cdots, d$ that the invariant holds for every node $\mathcal{N}_r$.

For the induction base, note that $\mathcal{N}_1$ is a leaf and thus $\mathcal{N}_1.value$ is written by the event $e^{\mathcal{L}}$ of Φ in line 17. Immediately after $e^{\mathcal{L}}$ the operation Φ is counted by $\mathcal{N}_1$. Since $e^{\mathcal{N}_1} = e^{\mathcal{L}}$ the invariant holds for $\mathcal{N}_1$.

For the induction step, assume that the invariant holds for $\mathcal{N}_r$. WLOG, $\mathcal{N}_r$ is the left child of $\mathcal{N}_{r+1}$. Let p_i be the process that executes Φ.

If either $cas_i^1(\mathcal{N}_{r+1},\Phi)$ or $cas_i^2(\mathcal{N}_{r+1},\Phi)$ (if it is ever issued) change the value of $\mathcal{N}_{r+1}$, then $\mathcal{N}_{r+1}$ starts counting Φ in $E_1^{\mathcal{N}_{r+1}} e^{\mathcal{N}_{r+1}}$ and the invariant holds. Hence, we consider the case in which both of these *CAS* events are trivial. This case occurs when $\mathcal{N}_{r+1}.value$ is changed by some process $p_j \neq p_i$ (executing a `WriteMax` operation Φ_j) between $read_i^1(\mathcal{N}_{r+1},\Phi)$ and $cas_i^1(\mathcal{N}_{r+1},\Phi)$, and by some process $p_k \neq p_i$ (executing a `WriteMax` operation Φ_k) between $read_i^2(\mathcal{N}_{r+1},\Phi)$ and $cas_i^2(\mathcal{N}_{r+1},\Phi)$.

WLOG, assume that the value of $\mathcal{N}_{r+1}.value$ is updated by $cas_j^1(\mathcal{N}_{r+1},\Phi_j)$ and $cas_k^1(\mathcal{N}_{r+1},\Phi_k)$, that $cas_j^1(\mathcal{N}_{r+1},\Phi_j) \overset{E}{\prec} cas_k^1(\mathcal{N}_{r+1},\Phi_k)$ and that $read\_lc_j^1(\mathcal{N}_{r+1},\Phi_j) \overset{E}{\prec} read\_lc_k^1(\mathcal{N}_{r+1},\Phi_k)$

If prior to $read\_lc_k^1(\mathcal{N}_{r+1},\Phi_k)$ the node $\mathcal{N}_r$ counts Φ then after $cas_k^1(\mathcal{N}_{r+1},\Phi_k)$, $\mathcal{N}_{r+1}$ starts counting Φ in $E_1^{\mathcal{N}_{r+1}} e^{\mathcal{N}_{r+1}}$ and the invariant holds.

The only case left to consider is that $\mathcal{N}_r$ does not count Φ in the execution prefix prior to $read\_lc_k^1(\mathcal{N}_{r+1},\Phi_k)$. We prove that this case is impossible. Assume towards a contradiction, that $read\_lc_k^1(\mathcal{N}_{r+1},\Phi_k)$ occurs before the event after which $\mathcal{N}_r$ counts Φ. Hence, by induction hypothesis, $read\_lc_k^1(\mathcal{N}_{r+1},\Phi_k) \overset{E}{\prec} e^{\mathcal{N}_r} \overset{E}{\prec} read_i^1(\mathcal{N}_{r+1},\Phi)$.

Since $read\_lc_k^1(\mathcal{N}_{r+1},\Phi_k) \overset{E}{\prec} cas_j^1(\mathcal{N}_{r+1},\Phi_j) \overset{E}{\prec} cas_k^1(\mathcal{N}_{r+1},\Phi_k)$, it is obvious that $p_j \neq p_k$. We denote the value stored in $\mathcal{N}_{r+1}.value$ as x upon $read_k^1(\mathcal{N}_{r+1},\Phi_k)$, as y upon $read_i^1(\mathcal{N}_{r+1},\Phi)$, as z upon $cas_i^1(\mathcal{N}_{r+1},\Phi)$ and as t upon $cas_k^1(\mathcal{N}_{r+1},\Phi_k)$. From Lemma 8, $x \leq y \leq z \leq t$. Since $cas_i^1(\mathcal{N}_{r+1},\Phi)$ does not update $\mathcal{N}_{r+1}.value$ then $y! = z$ and, consequently $x < t$. On the other hand, $cas_k^1(\mathcal{N}_{r+1},\Phi_k)$ is non-trivial, yielding $x = t$. This is a contradiction. □

LEMMA 10. *Let E' be a prefix of E and let $W(E') = \{\Phi_1, \cdots, \Phi_w\}$ denote the set of all `WriteMax` operations such that for $i = 1, \cdots, w$ the linearization point e_i of Φ_i appears in E'. If for each $i = 1, \cdots, w$ the operand of $\Phi_i \leq v$, then $T.root.value \leq v$ after E'.*

PROOF. Assume towards a contradiction, that $T.root.value = v' > v$ after E'. Obviously, the value v' is written by some `WriteMax` operation $\Phi \notin W(E')$ on some leaf $\mathcal{L}$ and then propagated to $T.root$. Let $\mathcal{N}_1, \cdots, \mathcal{N}_d$, where d is the depth of $\mathcal{L}$, denote the nodes in the propagation path of value v', such that $\mathcal{N}_1$ is $\mathcal{L}$ and $\mathcal{N}_d$ is $T.root$. For $r = 1, \cdots, d$, let $e^{\mathcal{N}_r}$ denote the event that writes the value v' to $\mathcal{N}_r$ such that $E' = E_1^{\mathcal{N}_r} e^{\mathcal{N}_r} E_2^{\mathcal{N}_r}$.

To obtain a contradiction we prove that $T.root$ counts Φ after $E_1^{\mathcal{N}_d} e^{\mathcal{N}_d}$ and thus $\Phi \in W(E')$. We consider this claim as a special case of the following invariant: for $r = 1, \cdots, d$, $\mathcal{N}_r$ counts Φ after $E_1^{\mathcal{N}_r} e^{\mathcal{N}_r}$. The proof is by induction on r.

For the induction base, we note that $\mathcal{N}_1$ is a leaf and thus $e^{\mathcal{N}_1}$ is a *write* issued by Φ. Hence, $\Phi \in \mathcal{C}(\mathcal{N}_1, E_1^{\mathcal{N}_1} e^{\mathcal{N}_1})$ and the invariant holds for $\mathcal{N}_1$.

For the induction step, assume that $\Phi \in \mathcal{C}(\mathcal{N}_r, E_1^{\mathcal{N}_r} e^{\mathcal{N}_r})$. Let Φ' be an operation by p_j that issues $e^{\mathcal{N}_{r+1}}$. WLOG, $\mathcal{N}_r$ is the left child of $\mathcal{N}_{r+1}$ and $e^{\mathcal{N}_{r+1}}$ is $cas_j^1(\mathcal{N}_{r+1},\Phi')$.

According to Algorithm A, $e^{\mathcal{N}_r} \overset{E}{\prec} read\_lc_j^1(\mathcal{N}_{r+1}, \Phi')$ $\overset{E}{\prec} cas_j^1(\mathcal{N}_{r+1}, \Phi') \equiv e^{\mathcal{N}_{r+1}}$. Since $\Phi \in \mathcal{C}(\mathcal{N}_r, E_1^{\mathcal{N}_r} e^{\mathcal{N}_r})$, from Definition 8, $\Phi \in \mathcal{C}(\mathcal{N}_{r+1}, E_1^{\mathcal{N}_{r+1}} e^{\mathcal{N}_{r+1}})$ and the invariant holds for $\mathcal{N}_{r+1}$. This is a contradiction. $\square$

LEMMA 11. *Let Φ be a* `WriteMax` *operation by p_i with operand v. Let event e be the* linearization point *of Φ in E. If $E = E'eE"$, then $T.root.value \geq v$ after $E'e$.*

PROOF. From Definition 9, $\Phi \in \mathcal{C}(T.root, E'e)$.

Let $\mathcal{L}$ be the leaf accessed by Φ. Let $\mathcal{N}_1, \cdots, \mathcal{N}_d$, where d is the depth of $\mathcal{L}$, denote the nodes in the path of Φ, such that $\mathcal{N}_1$ is $\mathcal{L}$ and $\mathcal{N}_d$ is $T.root$. For $r = 1, \cdots, d$, let $e^{\mathcal{N}_r}$ denote the first event in E such that $E' = E_1^{\mathcal{N}_r} e^{\mathcal{N}_r} E_2^{\mathcal{N}_r}$ and $\Phi \in \mathcal{C}(\mathcal{N}_r, E_1^{\mathcal{N}_r} e^{\mathcal{N}_r})$.

From Definition 9, $E'e \equiv E_1^{\mathcal{N}_d} e^{\mathcal{N}_d}$. We prove that after $E_1^{\mathcal{N}_d} e^{\mathcal{N}_d}$ the value of $T.root \geq v$, considering this as a special case of the following invariant: for $r = 1, \cdots, d$, $\mathcal{N}_r.value \geq v$ after $E_1^{\mathcal{N}_r} e^{\mathcal{N}_r}$. The proof is by induction on r.

For induction base, note that $\mathcal{N}_1$ is a leaf and thus $e^{\mathcal{N}_1}$ is the *write* of value v on $\mathcal{N}_1.value$ (in line 17). Hence $\mathcal{N}_1.value = v$ after $E_1^{\mathcal{N}_1} e^{\mathcal{N}_1}$ and the invariant holds for $\mathcal{N}_1$.

For induction step, assume that $\mathcal{N}_r.value \geq v$ after $E_1^{\mathcal{N}_r} e^{\mathcal{N}_r}$. Let Φ' be a `WriteMax` operation by p_j that issues $e^{\mathcal{N}_{r+1}}$. WLOG, $\mathcal{N}_r$ is the left child of $\mathcal{N}_{r+1}$ and $e^{\mathcal{N}_{r+1}}$ is $cas_j^1(\mathcal{N}_{r+1}, \Phi')$. We consider the following cases.

1. if $\Phi = \Phi'$, then by Invariant 1, $\mathcal{N}_r$ counts Φ in execution prior to $read\_lc_i^1(\mathcal{N}_{r+1}, \Phi) \equiv read\_lc_j^1(\mathcal{N}_{r+1}, \Phi')$.
2. if $\Phi \neq \Phi'$, then obviously $p_i \neq p_j$ and from Definition 9, $\mathcal{N}_r$ counts Φ in the execution prefix prior to $read\_lc_j^1(\mathcal{N}_{r+1}, \Phi')$.

According to Algorithm A, the value v' written by $cas_j^1(\mathcal{N}_{r+1}, \Phi') \equiv e^{\mathcal{N}_{r+1}}$ is greater than or equal to the value read by $read\_lc_j^1(\mathcal{N}_{r+1}, \Phi')$. Since $e^{\mathcal{N}_r} \overset{E}{\prec} read\_lc_j^1(\mathcal{N}_{r+1}, \Phi')$, $\mathcal{N}_r.value \geq v$ prior to $read\_lc_j^1(\mathcal{N}_{r+1}, \Phi')$ and thus $v' \geq v$. Consequently, $\mathcal{N}_{r+1}.value \geq v$ after $E_1^{\mathcal{N}_{r+1}} e^{\mathcal{N}_{r+1}}$ and the invariant holds for $\mathcal{N}_{r+1}$. $\square$

LEMMA 12. *Let Φ be a* `ReadMax` *operation in H, such that $H = H_1 \Phi H_2$. Then Φ returns the largest value written in H_1 (or $-\infty$, if no* `WriteMax` *appears in H_1).*

PROOF. Let e be the linearization point of Φ in E, such that $E = E_1 e E_2$. From Definition 9, e is a *read* of $T.root.value$ and according to Algorithm A, Φ returns the value obtained by e.

If no `WriteMax` appears in H_1, then no event is visible on $T.root.value$ after E_1. Hence e reads the initial value of $T.root.value$, which is $-\infty$.

Let v be the largest value written in H_1 and let Φ_w be the `WriteMax` operation that wrote v. We prove that $T.root.value = v$ after E_1. Let e_w be the *linearization point* of Φ_w such that $E_1 = E_1' e_w E_2'$. From Lemma 11, $T.root.value \geq v$ after $E_1' e_w$ and thus after E_1. Moreover, from Lemma 10, $T.root.value \leq v$ after E_1, otherwise v was not the largest value written in H_1. $\square$

From Observations 1 - 2 and from Lemmas 7 - 9, for every operation Φ in E its *linearization point* is uniquely defined within the time interval of Φ, and thus H preserves a partial precedence order of operations in E. Moreover, from Lemma 12, H is a legal sequential history of *maxRegister*. This immediately leads us to the following theorem.

THEOREM 5. *Algorithm A is a linearizable implementation of N-process unbounded* maxRegister.

Wait-freedom and Complexity.

Clearly, Algorithm A is wait-free. The number of events issued by `ReadMax` is constant. The number of events issued by `WriteMax`(v) is proportional to the depth d of leaf $\mathcal{L}$:

1. if $v < N$, then $\mathcal{L} \in T_L$ and from the definition of B_1 tree, $d = O(\log v) = O(\min(\log N, \log v))$.
2. if $v \geq N$, then $\mathcal{L} \in T_R$ and from the definition of complete binary tree, $d = O(\log N) = O(\min(\log N, \log v))$

THEOREM 6. *Algorithm A is wait-free; the step-complexity of the* `ReadMax` *operation is $O(1)$ and the step-complexity of the* `WriteMax`(v) *is $O(\min(\log N, \log v))$.*

6. REFERENCES

[1] J. H. Anderson and Y.-J. Kim. An improved lower bound for the time complexity of mutual exclusion. *Distributed Computing*, 15(4):221–253, 2002.

[2] J. Aspnes, H. Attiya, and K. Censor. Polylogarithmic concurrent data structures from monotone circuits. *J. ACM*, 59(1), Feb. 2012. Previous version in *PODC*, pages 36–45, 2009.

[3] J. Aspnes, H. Attiya, K. Censor-Hillel, and F. Ellen. Faster than optimal snapshots (for a while): preliminary version. In *PODC*, pages 375–384, 2012.

[4] J. Aspnes, H. Attiya, K. Censor-Hillel, and D. Hendler. Lower bounds for restricted-use objects: extended abstract. In *SPAA*, pages 172–181, 2012.

[5] J. Aspnes and K. Censor. Approximate shared-memory counting despite a strong adversary. *ACM Transactions on Algorithms*, 6(2), 2010.

[6] H. Attiya and D. Hendler. Time and space lower bounds for implementations using k-cas. *IEEE Trans. Parallel Distrib. Syst.*, 21(2):162–173, 2010.

[7] M. A. Bender and S. Gilbert. Mutual exclusion with $O(\log \log n)$ amortized work. In *FOCS*, pages 728–737, 2011.

[8] J. L. Bentley and A. C. chih Yao. An almost optimal algorithm for unbounded searching, 1975.

[9] B. Bollobas. *Extremal Graph Theory*. Dover Publications, Incorporated, 2004.

[10] M. Herlihy. Wait-free synchronization. *ACM Trans. Prog. Lang. Syst.*, 13(1):124–149, January 1991.

[11] M. Herlihy, V. Luchangco, and M. Moir. Obstruction-free synchronization: Double-ended queues as an example. In *Proceedings of the 23rd IEEE International Conference on Distributed Computing Systems (ICDCS'03)*, pages 522–529, 2003.

[12] M. Herlihy and J. M. Wing. Linearizability: a correctness condition for concurrent objects. *ACM Trans. Prog. Lang. Syst.*, 12(3):463–492, June 1990.

[13] P. Jayanti. A time complexity lower bound for randomized implementations of some shared objects. In *PODC*, pages 201–210, 1998.

[14] P. Jayanti. *f*-arrays: implementation and applications. In *PODC*, pages 270–279, 2002.

[15] Y.-J. Kim and J. H. Anderson. A time complexity bound for adaptive mutual exclusion. In *DISC*, pages 1–15, 2001.