Lower Bounds for Restricted-Use Objects

(Extended Abstract)

James Aspnes[*] Hagit Attiya[†] Keren Censor-Hillel[‡] Danny Hendler[§]

February 2, 2012

Abstract

Concurrent objects play a key role in the design of applications for multi-core architectures, making it imperative to precisely understand their complexity requirements. For some objects, it is known that implementations can be significantly more efficient when their usage is restricted. However, apart from the specific restriction of one-shot implementations, where each process may apply only a single operation to the object, very little is known about the complexities of objects under general restrictions.

This paper draws a more complete picture by defining a large class of objects for which an operation applied to the object can be "perturbed" L consecutive times, and proving lower bounds on the time and space complexity of deterministic implementations of such objects. This class includes bounded-value max registers, limited-use approximate and exact counters, and limited-use collect and compare-and-swap objects; L depends on the number of times the object can be accessed or the maximum value it can support.

For implementations that use only historyless primitives, we prove lower bounds of $\Omega(\min(\log L, n))$ on the worst-case step complexity of an operation, where n is the number of processes; we also prove lower bounds of $\Omega(\min(L, n))$ on the space complexity of these objects. When arbitrary primitives can be used, we prove that either some operation incurs $\Omega(\min(\log L, n))$ memory stalls or some operation performs $\Omega(\min(\log L, n))$ steps.

In addition to these deterministic lower bounds, the paper establishes a lower bound on the expected step complexity of restricted-use randomized approximate counting in a weak oblivious adversary model.

[*]Department of Computer Science, Yale University
[†]Department of Computer Science, Technion
[‡]Computer Science and Artificial Intelligence Laboratory, MIT
[§]Department of Computer Science, Ben-Gurion university of the Negev

1 Introduction

With multi-core and multi-processor systems now prevalent, there is growing need to gain better understanding of concurrent objects and, specifically, to establish lower bounds on the cost of implementing them. An important general class of concurrent objects, defined by Jayanti, Tan and Toueg [15], are *perturbable* objects, including widely-used objects, such as *counters*, *max registers*, *compare-and-swap*, *single-writer snapshot* and *fetch-and-add*.

Lower bounds are known for *long-lived* implementations of perturbable objects, where processes apply an unbounded number of operations to the object. For example, Jayanti *et al.* [15] consider obstruction-free implementations of perturbable objects from *historyless* primitives, such as *read*, *write*, *test-and-set* and *swap*. They prove that such implementations require $\Omega(n)$ space and that the worst-case step-complexity of the operations they support is $\Omega(n)$, where n is the number of processes sharing the object.

In some applications, however, objects are used in a restricted manner. For example, there might be a bound on the total number of operations applied on the object, or a bound on the values that the object needs to support. When an object is designed to allow only restricted use, it is sometimes possible to construct more efficient implementations than for the general case.

Indeed, Aspnes, Attiya and Censor-Hillel [3] showed that at least some restricted-use perturbable objects admit implementations that "beat" the lower bound of [15]. For example, a max register can do a write of v in $O(\min(\log v, n))$ steps, while a counter limited to m increments can do each increment in $O(\min(\log^2 m, n))$ steps. Such a restricted-use counter leads to a randomized consensus algorithm with $O(n)$ individual step complexity [4], while restricted-use counters and max registers are used in a mutual exclusion algorithm with sub-logarithmic amortized work [7].

This raises the natural question of determining lower bounds on the complexity of restricted-use objects. The proof of Jayanti *et al.* [15] breaks for restricted-use objects because the executions constructed by these proofs exceed the restrictions on these objects.

For the specific restriction of *one-time* object implementations, where each process applies exactly one operation to the object, there are lower bounds which are logarithmic in the number of processes, for specific objects [1,2,6] and generic perturbable objects [14]. Yet, these techniques yield bounds that are far from the upper bounds, e.g., when the object can be perturbed a super-polynomial number of times.

This paper provides a more complete picture of the cost of implementing restricted-use objects by studying the middle ground. We give time and space lower bounds for implementations of objects that are only required to work under restricted usage, for general families of restrictions.

We define the notion of *L-perturbable objects* that strictly generalizes classical perturbability; specific examples are bounded-value max registers, limited-use approximate and exact counters, and limited-use compare-and-swap and collect objects.[1] L, the perturbation bound, depends on the number of times the object can be accessed or the maximum value it can support (see Table 1).

For L-perturbable objects, we show lower bounds on the step and space complexity of obstruction-free deterministic implementations from historyless primitives. The step complexity lower bound is $\Omega(\min(\log L, n))$, and its proof employs a technique that we call *backtracking covering*, introduced by Fich, Hendler and Shavit in [11] and later used in [5]. The space complexity lower bound is $\Omega(\min(L, n))$.

[1] A single-writer snapshot object is also a collect object (the converse is, in general, false). Therefore, our lower bounds for the collect object also hold for the single-writer snapshot object.

	perturbation bound (L)	step complexity	max(steps, stalls)	space complexity	rand. step complexity
compare & swap	$\sqrt[3]{m}-1$	$\Omega(\min(\log m, n))$	$\Omega(\min(\log m, n))$	$\Omega(\min(\sqrt[3]{m}, n))$	—
collect	$m-1$	$\Omega(\min(\log m, n))$	$\Omega(\min(\log m, n))$	$\Omega(\min(m, n))$	—
max register	$m-1$	$\Omega(\min(\log m, n))$ (also [3])	$\Omega(\min(\log m, n))$	$\Omega(\min(m, n))$	$\Omega(\frac{\log\log m}{\log\log\log m})$ (for $m \leq n$, [3])
k-additive counter	$\sqrt{\frac{m}{k}}-1$	$\Omega(\min(\log \frac{m}{k}, n))$ (also [3])	$\Omega(\min(\log \frac{m}{k}, n))$	$\Omega(\min(\sqrt[k]{m}, n))$	$\Omega(\frac{\log\log m}{\log\log\log m})$ (for $m \leq n$)

Table 1: Summary of lower bounds for restricted-use objects; where m is the maximum value assumed by the object or the bound on the number of operations applied to it. All the bounds are derived in this paper, except when stated otherwise.

We also consider implementations that can apply *arbitrary* primitives and not only historyless primitives, and use the *memory stalls* measure [8] to quantify the contention incurred by such implementations. We use backtracking covering to prove that either an implementation's worst-case operation step complexity is $\Omega(\min(\log L, n))$ or some operation incurs $\Omega(\min(\log L, n))$ stalls.

In addition to our deterministic lower bounds, we establish a lower bound of $\Omega(\log\log m/\log\log\log m)$ on the expected step complexity of *randomized* m-valued *c-accurate counters*, a particularly weak class of counters that allow a multiplicative error of factor at most c. Our lower bound employs Yao's Principle [16] and assumes a weak oblivious adversary.

Table 1 summarizes the lower bounds for specific L-perturbable objects.

Aspnes *et al.* [3] prove lower bounds on obstruction-free implementations of max registers and approximate counters from historyless primitives: an $\Omega(\min(\log m, n))$ step lower bound on deterministic and $\Omega(\log m/\log\log m)$ lower bound, for $m \leq n$, on the expected step complexity of randomized implementations. These bounds, however, use a different proof technique, which is specifically tailored for the semantics of the particular objects, and does not seem to generalize to the restricted-use versions of *arbitrary* perturbable objects. Moreover, they neither prove space-complexity lower bounds nor consider implementations from arbitrary primitives.

2 Model and Definitions

A shared-memory system consists of n *asynchronous* processes $p_1, \ldots, p_n$ communicating by applying primitive operations (*primitives*) on shared *base objects*. An application of each such primitive is a shared memory *event*. A *step* taken by a process consists of local computation followed by one shared memory event.

A primitive is *nontrivial* if it may change the value of the base object to which it is applied, e.g., a *write* or a *read-modify-write*, and *trivial* otherwise, e.g., a *read*. Let o be a base object that is accessed with two primitives f and f'; f *overwrites* f' on o [10], if starting from any value v of o, applying f' and then f results in the same value as applying just f, using the same input parameters (if any) in both cases. A set of primitives is *historyless* if all the nontrivial primitives in the set overwrite each other; we also require that each such primitive overwrites itself. A set that includes the write and swap primitives is an example of a historyless set of primitives.

Executions and Operations: An *execution fragment* is a sequence of shared memory events applied by processes. An execution fragment is p_i-free if it contains no steps of process p_i. An *execution* is an execution fragment that starts from an initial configuration (in which all shared variables and processes' local states assume their initial values).

An *operation instance* of an operation *Op* on an implemented object is a subsequence of an execution, in which some process p_i performs the operation *Op* on the object. The primitives applied by the operation instance may depend on the values of the shared base objects before this operation instance starts and during its execution (p_i's steps may be interleaved with steps of other processes). Most implementations are required to be *linearizable* [13]. An implementation is *obstruction-free* [12] if a process terminates its operation instance if it runs in isolation long enough.

A process p is *active* after execution α if p is in the middle of performing an operation instance, i.e., p has applied at least one event of the operation instance in α, but the instance is not complete in α. Let *active(α)* denote the set of processes that are active after α. If p is not active after α, we say that p is *idle* after α. A base object o is *covered after* an execution α if there is a process p in the configuration resulting from α that has a nontrivial event about to access o; we say that p *covers* o after α.

Restricted-Use Objects: Our main focus in this paper is on objects that support restricted usage. One example of such objects are objects that have a limit on the number of operation instances that can be performed on them, as captured by the following definition. An *m-limited-use* object is an object that allows at most m operation instances; m is the *limit* of the object.

Another type of objects with restricted usage are objects that have a value associated with their state which cannot exceed some bound. Examples are bounded max-registers and bounded counters [3], whose definitions appear in the appendix.

We also consider collect and compare-and-swap objects. A *collect* object provides two operations: a *store*(*val*) by process p_i sets *val* to the latest value for p_i. A *collect* operation *cop* returns a *view*, $\langle v_1, \ldots, v_n \rangle$, satisfying the following properties: 1) if $v_j = \perp$, then no *store* operation by p_j completes before *cop* starts, and 2) if $v_j \neq \perp$, then v_j is the operand of a *store* operation *sop* by p_j that starts before *cop* completes and there is no store operation by p_j that starts after *sop* completes and completes before *cop* starts. A linearizable b-valued *compare-and-swap* object assumes values from $\{1, \ldots, b\}$ and supports the operations *read* and *CAS(u,v)*, for all $u, v \in \{1, \ldots, b\}$. When the object's value is u, *CAS(u,v)* changes its value to v and returns *true*; when the object's value differs from u, *CAS(u,v)* returns *false* and does not change the object's value.

3 Lower Bounds for Deterministic Restricted-Use Objects

In this section, we prove lower bounds for obstruction-free implementations of some restricted-use objects. Our starting point is the definition of *perturbable* objects by Jayanti *et al.* [15]. Roughly speaking, an object is perturbable if in some class of executions, events applied by an operation of one process influence the response of an operation of another process. The flavor of the argument used by Jayanti *et al.* to obtain their linear lower bound is that since the perturbed operation needs to return different responses with each perturbation, it must be able to distinguish between perturbed executions, implying that it must perform an increasing number of accesses to base objects.

Following is the formal definition of perturbable objects.

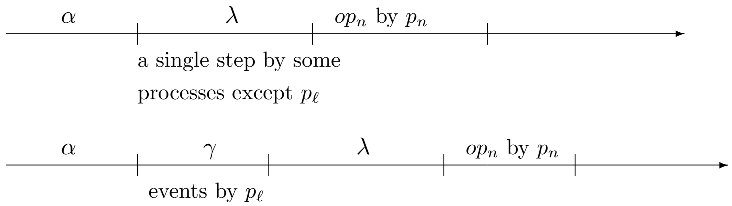

Figure 1: A perturbable object: op_n returns different responses in the two executions.

Definition 1 *(See Figure 1.) An object $\mathcal{O}$ is* perturbable *if there is an operation instance op_n by process p_n, such that for any p_n-free execution $\alpha\lambda$ where no process applies more than a single event in λ and for some process $p_\ell \neq p_n$ that applies no event in λ, there is an extension of α, γ, consisting of events by p_ℓ, such that p_n returns different responses when performing op_n by itself after $\alpha\lambda$ and after $\alpha\gamma\lambda$.*

We observe that $\alpha\gamma\lambda$ in the above definition is an execution, since no process applies more than a single event in λ and p_ℓ applies no events in λ.

The linear lower bounds [15] on the space and step complexity of obstruction-free implementations on perturbable objects (as defined in Definition 1 above) are obtained by constructing executions of unbounded length, hence they do not apply in general for restricted-use objects.

To prove lower bounds for restricted-use objects, we define a class of *L-perturbable* objects. As opposed to the definition of a perturbable object, we do not require every execution of an L-perturbable object to be perturbable, since this requirement is in general not satisfied by restricted-use objects. For such objects, some executions already reach the limit or bound of the object, not allowing any further operation to affect the object, which rules out a perturbation of these executions. To achieve our lower bounds we only need to show the existence of a special perturbing sequence of executions rather than attempting to perturb any execution. The longer the sequence, the higher the lower bound, since the perturbed operation will have to access more base objects in order to distinguish between executions in the sequence and be able to return different responses.

Definition 2 *An object O is L-*perturbable *if there exists an operation instance op_n by p_n such that an L-perturbing execution of O can be constructed as follows: The empty execution is 0-perturbing. Assume the object has a $(k-1)$-perturbing execution $\alpha_{k-1}\lambda_{k-1}$, where no process applies more than a single event in λ_{k-1}.*

1. *If $|\lambda_{k-1}| = n - 1$, then we say that $\alpha_{k-1}\lambda_{k-1}$ is* saturated*, and the execution $\alpha_k\lambda_k$ with $\alpha_k = \alpha_{k-1}$, $\lambda_k = \lambda_{k-1}$ is k-*perturbing.

2. *Otherwise, if there exists a process $p_\ell \neq p_n$ that applies no event in λ_{k-1} and an extension of α_{k-1}, γ, consisting of events by p_ℓ, such that p_n returns different responses when performing op_n by itself after $\alpha_{k-1}\lambda_{k-1}$ and after $\alpha_{k-1}\gamma\lambda_{k-1}$, then we define a k-perturbing execution as follows. Let $\gamma = \gamma' e \gamma''$, where e is the first event of γ such that op_n returns different responses after $\alpha_{k-1}\lambda_{k-1}$ and after $\alpha_{k-1}\gamma' e \lambda_{k-1}$. Let λ be any permutation of the events in λ_{k-1} and the event e, and let λ', λ'' be any two sequences of events such that $\lambda = \lambda'\lambda''$. The execution $\alpha_k\lambda_k$ is k-*perturbing*, where $\alpha_k = \alpha_{k-1}\gamma'\lambda'$ and $\lambda_k = \lambda''$.*

If an object is L-perturbable, then, starting from the initial configuration, we may construct a sequence of $L + 1$ perturbing executions, $\alpha_k\lambda_k$, for $0 \leq k \leq L$, each of which extending its predecessor perturbing execution. If for some i, $\alpha_i\lambda_i$ is saturated, then we cannot further extend the sequence of perturbing executions since we do not have available processes to perform the perturbation. However, in this case we have lower bounds that are linear in n. For presentation simplicity, we assume in this case that the rest of the sequence's perturbing executions are identical to $\alpha_i\lambda_i$.

Definition 2 allows flexibility in determining which of the events of λ_{k-1} are contained in λ_k and which are contained in α_k. We use this flexibility to prove lower bounds on the step, space and stall-complexity of L-perturbable objects.

The definition implies that every perturbable object is L-perturbable for every integer $L \geq 0$, hence, the class of L-perturbable objects generalizes the class of perturbable objects. On the other hand, there are L-perturbable objects that are not perturbable; for example, a b-bounded n-process max register, for $b \in \mathrm{poly}(n)$, is not perturbable in general, by the algorithm of [3]. That is, the class of perturbable objects is a proper subset of the class of L-perturbable objects.

Lemma 5 (in the appendix) establishes that several common restricted-use objects are L-perturbable, where L is a function of the limit on the number of different operations that may be applied to them. The challenge in proving this lemma is in increasing L, which later translates to higher lower bounds. The specific bounds are summarized in Table 1.

3.1 Lower bounds for implementations using historyless objects

We define the concept of an access-perturbation sequence, and prove a step-complexity lower bound for objects that admit such a sequence.

Definition 3 *(See Figure 2.) An* access-perturbation sequence of length L *of an operation instance* op_n *by process* p_n *on an object* $\mathcal{O}$ *is a sequence of executions* $\{\alpha_r\lambda_r\phi_r\}_{r=0}^{L}$, *such that* $\alpha_0\lambda_0$ *is empty,* ϕ_0 *is an execution of* op_n *by* p_n *starting from the initial configuration, and for every* r, $1 \leq r \leq L$, *the following properties hold:*

1. *The execution* $\alpha_r\lambda_r$ *is* p_n*-free.*
2. *In* ϕ_r, *process* p_n *runs solo after* $\alpha_r\lambda_r$ *until it completes the operation instance* op_n, *in the course of which it accesses the base objects* $B_r^1, \ldots, B_r^{i_r}$.
3. λ_r *consists of* $j_r \geq 0$ *nontrivial events applied by* j_r *distinct processes,* $p_r^1, \ldots, p_r^{j_r}$ *to distinct base objects* $O_r^1, \ldots, O_r^{j_r}$, *respectively, all of which are accessed by* p_n *in* ϕ_r. *If* $j_r = n-1$, *we say that* $\alpha_r\lambda_r\phi_r$ *is* saturated.
4. (a) *If* $\alpha_{r-1}\lambda_{r-1}\phi_{r-1}$ *is saturated, then we let* $\alpha_r = \alpha_{r-1}$, $\lambda_r = \lambda_{r-1}$ *and* $\phi_r = \phi_{r-1}$.
   (b) *Otherwise, we let* $\alpha_r = \alpha_{r-1}\gamma'_r\lambda'_{r-1}$, *and* $\lambda_r = \lambda''_{r-1}e_r$, *where* λ'_{r-1} *is the subset of* λ_{r-1} *containing all events to base objects that are not accessed by* p_n *in* ϕ_r, λ''_{r-1} *is the subset of* λ_{r-1} *containing all events to base objects that are accessed by* p_n *in* ϕ_r, *and* $\gamma'_r e_r$ *is an execution fragment by a process* p_{ℓ_r} *not taking steps in* λ_{r-1}, *where* e_r *is its first nontrivial event to a base object in* $\{B_{r-1}^1, \ldots, B_{r-1}^{i_{r-1}}\} \setminus \{O_{r-1}^1, \ldots, O_{r-1}^{j_{r-1}}\}$.

Next, we prove a step lower bound for implementations that have an access-perturbation sequence. If the sequence is saturated, then the lower bound is linear in the number of processes, otherwise it is logarithmic in the length of the sequence. Our goal is to prove that p_n has to access

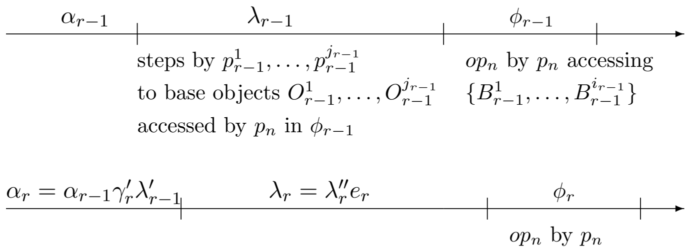

Figure 2: An access-perturbation sequence of length L: the above describes the executions for every r, $1 \leq r \leq L$. Notice that $\alpha_r\lambda_r$ is p_n-free for every r.

a large number of base objects as it runs solo while performing an instance op_n of Op in one of the executions of op_n's access-perturbation sequence. Let π_r denote the sequence of base objects accessed by p_n in ϕ_r, in the order of their first access in ϕ_r; π_r is p_n's *solo path* in ϕ_r. If all the objects accessed in λ_{r-1} are also in λ_r, i.e., p_n accesses them also in ϕ_r, then $\lambda_r = \lambda_{r-1}e_r$. However, the application of e_r may have the undesirable effect (from the perspective of an adversary) of making π_r shorter than π_{r-1}: p_n may read the information written by p_{ℓ_r} and avoid accessing some other objects that were previously in π_{r-1}.

To overcome this difficulty, we employ the *backtracking covering* technique [5, 11]. The observation underlying this technique is that objects that are in π_{r-1} will be absent from π_r only if the additional object to which p_{ℓ_r} applies the nontrivial event e_r precedes them in π_{r-1}. Thus the set of objects along π_r that are covered after $\alpha_r\lambda_r$ is 'closer', in a sense, to the beginning of p_n's solo path in ϕ_{r-1}. It follows that if there are many access-perturbation sequence executions r for which $|\pi_r| < |\pi_{r-1}|$, then one of the solo paths π_r must be 'long'.

To capture this intuition, we define Ψ, a monotonically-increasing progress function of r. Ψ_r is a $(\log L)$-digit binary number defined as follows. Bit 0 (the most significant bit) of Ψ_r is 1 if and only if the first object in π_r is covered after α_r (by one of the events of λ_r); bit 1 of Ψ_r is 1 if and only if the second object in π_r exists and is covered after α_r, and so on. Note that we do not need to consider paths that are longer than $\log L$. If such a path exists, the lower bound clearly holds.

To construct the r'th access-perturbation sequence execution, we deploy a free process, p_{ℓ_r} and let it run solo until it is about to write to an uncovered object, O, along π_r. (Since the sequence is not saturated, it follows from Property 4(b) of Definition 3 that such p_{ℓ_r} and O exist.) In terms of Ψ, this implies that the covering event e_r might flip some of the digits of Ψ_{r-1} from 1 to 0. But O corresponds to a more significant digit, and this digit is flipped from 0 to 1, hence $\Psi_r > \Psi_{r-1}$ must hold. Thus we can construct executions $\alpha_r\lambda_r\phi_r$, for $1 \leq r \leq L$, in each of which Ψ_r increases. It follows that $\Psi_r = L - 1$ must eventually hold, implying that π_r's length is $\Omega(\log L)$.

Theorem 1 *Let A be an n-process obstruction-free implementation of an L-perturbable object $\mathcal{O}$ from historyless primitives. Then A has an execution in which some process accesses $\Omega(\min(\log L, n))$ distinct base objects during a single operation instance.*

Proof: Lemma 6 (in the appendix) shows that any implementation of O from historyless primitives has an access-perturbation sequence of length $L \geq 1$, $\{\alpha_r\lambda_r\phi_r\}_{r=0}^{L}$. If the sequence is saturated, then Definition 3 immediately implies that p_n accesses $n-1$ distinct base objects in the course of performing ϕ_r, and the lower bound holds. Otherwise, we show that op_n accesses $\Omega(\log L)$ distinct base objects in one of these executions.

Let $\pi_r = B^1_r \ldots B^{i_r}_r$ denote the sequence of all distinct base objects accessed by p_n in ϕ_r (after $\alpha_r\lambda_r$) according to Property 2 of Definition 3, and let S_{π_r} denote the set of these base objects. Let

$S_r^C = \{O_r^1, \ldots, O_r^{j_r}\}$ be the set of base objects defined in Property 3 of Definition 3. Observe that, by Property 3, $S_r^C \subseteq S_{\pi_r}$ holds. Without loss of generality, assume that $O_r^1, \ldots, O_r^{j_r}$ occur in π_r in the order of their superscripts.

In the execution $\alpha_r \lambda_r \phi_r$, p_n accesses i_r distinct base objects. Thus, it suffices to show that some i_r is in $\Omega(\log L)$. For $j \in \{1, \ldots, i_r\}$, let b_r^j be the indicator variable whose value is 1 if $B_r^j \in S_r^C$ and 0 otherwise. We associate an integral progress parameter, Ψ_r, with each $r \geq 0$, defined as follows:

$$\Psi_r = \sum_{j=1}^{i_r} b_r^j \cdot \frac{L}{2^j} \ .$$

For simplicity of presentation, and without loss of generality, assume that $L = 2^s$ for some integer $s > 0$, so $s = \log L$. If $i_r > s$ for some r then we are done. Assume otherwise, then Ψ_r can be viewed as a binary number with s digits whose j'th most significant bit is 1 if the j'th base object in π_r exists and is in S_r^C, or 0 otherwise. This implies that the number of 1-bits in Ψ_r equals $|S_r^C|$. Our execution is constructed so that Ψ_r is monotonically increasing in r and eventually, for some r', $\Psi_{r'}$ equals $L - 1 = L \sum_{j=1}^{s} \frac{1}{2^j}$. This would imply that p_n accesses exactly s base objects during $\phi_{r'}$ (after $\alpha_{r'} \lambda_{r'}$).

We next show that $\Psi_r > \Psi_{r-1}$, for every $0 < r \leq L$. Since $\alpha_{r-1}\lambda_{r-1}\phi_{r-1}$ is not saturated, by Property 4(b) of Definition 3, there is a process p_{ℓ_r} that takes no steps in λ_{r-1}, and an execution fragment $\gamma_r' e_r$ of p_{ℓ_r} after α_{r-1}, such that e_r is the first nontrivial event of p_{ℓ_r} in $\gamma_r' e_r$ to a base object in $\{B_{r-1}^1, \ldots, B_{r-1}^{i_{r-1}}\} \setminus \{O_{r-1}^1, \ldots, O_{r-1}^{j_{r-1}}\}$. By Property 2 of that definition, this object is accessed by p_n in ϕ_r. Let k be the index of the object among the objects accessed in ϕ_{r-1}, i.e., it is B_{r-1}^k. This implies that $B_{r-1}^k \in S_{\pi_{r-1}} \setminus S_{r-1}^C$.

As $B_{r-1}^k \notin S_{r-1}^C$, we have $b_{r-1}^k = 0$. Since e_r is the first nontrivial event of p_{ℓ_r} in $\gamma_r' e_r$ to a base object in $S_{\pi_{r-1}} \setminus S_{r-1}^C$, we have that the values of objects $B_{r-1}^1 \cdots B_{r-1}^{k-1}$ are the same after $\alpha_{r-1}\lambda_{r-1}$ and $\alpha_r \lambda_r$. It follows that $b_{r-1}^j = b_r^j$ for $j \in \{1, \ldots, k-1\}$. This implies, in turn, that $B_{r-1}^k = B_r^k$. As $B_r^k \in S_r^C$, we have $b_r^k = 1$. In the appendix, we use the observation that $b_{r-1}^k = 0$ to prove that $\Psi_r > \Psi_{r-1}$ (Lemma 7). Since $\Psi_0 = 0$ and since Ψ_r strictly grows with r and can never exceed $L - 1$, it follows that $\Psi_L = L - 1$, which concludes the proof. $\square$

The specific lower bounds appear in Theorem 8 in the appendix, and are summarized in Table 1.

To prove space-complexity lower bounds on L-perturbable objects, we construct perturbing sequences in which many objects are covered; not all of them are necessarily accessed by the reader, but, nevertheless, they must be distinct, giving a lower bound on the number of base objects. In the appendix, we define a cover-perturbation sequence, which immediately yields a space-complexity lower bound for objects that admit such a sequence, linear in its length (Theorem 9), and prove that every L-perturbable object has such a sequence (Lemma 10). Together with Lemma 5, this gives the specific lower bounds stated in Theorem 11 (in the appendix) and summarized in Table 1.

3.2 Lower bounds for implementations using arbitrary primitives

The number of steps performed by an operation, as we have measured for implementations using only historyless objects, is not the only factor influencing the performance of an operation. The performance of a concurrent object implementation is also influenced by the extent to which multiple processes *simultaneously* access widely-shared memory locations. Dwork *et al.* [8] introduced a formal model to capture such contention, taking into consideration both the number of steps taken by a process and the number of *stalls* it incurs as a result of memory contention with other processes. More formally, an event e applied by a process p to object O in an execution α *incurs k memory stalls* if it is immediately preceded by k events by distinct processes different than p that apply nontrivial primitives to O.

Our next result shows a lower bound on implementations using *arbitrary* read-modify-write primitives. Its proof employs a variation of the backtracking covering technique, similar to the proof of Theorem 1. The earlier proof uses access-perturbable sequence of executions, in which each new execution deploys a process to cover an object that is not covered in the preceding execution. Such a series of executions cannot, in general, be constructed for algorithms that may use arbitrary primitives. Instead, the proof constructs a series of executions in which each new execution deploys a process that covers *some* object along p_n's path.

Definition 4 *An* access-stall perturbation sequence *of length L of an operation instance op_n by process p_n on an object $\mathcal{O}$ is a sequence of executions $\alpha_r \sigma_{r,1} \cdots \sigma_{r,j_r} \rho_r$, such that α_0 is empty, $j_0 = 0$, ρ_0 is an execution of op_n by p_n starting from the initial configuration, and for every r, $1 \leq r \leq L$, the following properties hold:*

1. *α_r is p_n-free,*

2. *in ρ_r process p_n runs solo until it completes the operation instance op_n; in this instance, p_n accesses the base objects $B_r^1, \ldots, B_r^{i_r}$,*

3. *there is a subsequence $O_r^1, \ldots O_r^{j_r}$ of disjoint objects in $B_r^1, \ldots B_r^{i_r}$ and disjoint nonempty sets of processes $S_r^1, \ldots, S_r^{j_r}$ such that, for $j = 1, \ldots, j_r$,*
   - *each process in S_r^j covers O_r^j after α_r, and*
   - *in $\sigma_{r,j}$, process p_n applies events until it is about to access O_r^j for the first time, then each of the processes in S_r^j accesses O_r^j, and, finally, p_n accesses O_r^j.*

4. *let λ_{r-1} be the subsequence of events by the processes in $S_{r-1}^1 \cup \cdots S_{r-1}^{j_{r-1}}$ that are applied in $\sigma_{r-1,1} \cdots \sigma_{r-1,j_{r-1}}$, then $\alpha_{r-1}\lambda_{r-1}$ is an $r-1$-perturbing execution; if $\alpha_{r-1}\lambda_{r-1}$ is saturated, then we say that $\alpha_{r-1}\sigma_{r-1,1} \cdots \sigma_{r-1,j_{r-1}}\rho_{r-1}$ is saturated,*

5. *If $\alpha_{r-1}\sigma_{r-1,1} \cdots \sigma_{r-1,j_{r-1}}\rho_{r-1}$ is saturated, then the r'th execution in the access-stall perturbation sequence is defined as identical to it. Otherwise, the following holds: $O_r^{j_r} = B_{r-1}^k$, for some $1 \leq k \leq i_{r-1}$; $B_r^i = B_{r-1}^i$, for all $i \in \{1, \ldots k\}$; $O_{r-1}^i = O_r^i$ and $S_{r-1}^i = S_r^i$ for all objects O_{r-1}^i that precede B_{r-1}^k in the sequence $B_{r-1}^1, \ldots, B_{r-1}^{i_r}$; and either $B_{r-1}^k \notin \{O_{r-1}^1, \ldots, O_{r-1}^{j_{r-1}}\}$ or $O_r^{j_r} = O_{r-1}^{j_r}$ and $|S_r^{j_r}| = |S_{r-1}^{j_r}| + 1$.*

Theorem 2 *Let A be an n-process obstruction-free implementation of an L-perturbable object $\mathcal{O}$ from any read-modify-write primitives. Then A has an execution in which some process either accesses $\Omega(\min(\log L, n))$ distinct base objects or incurs $\Omega(\min(\log L, n))$ memory stalls, during a single operation instance.*

Proof: For simplicity and without loss of generality, assume that $L = 2^{2s}$ for some integer s. If A has an execution in which some process accesses s distinct base objects during a single operation instance, then the theorem holds. Otherwise, it can be shown that A has an access-stall perturbation sequence of length L (Lemma 12 in the appendix). If one of these executions, $\alpha_r\sigma_{r,1}\cdots\sigma_{r,j_r}\rho_r$, for some $r \le L$, is saturated, then it follows from Definition 4 that p_n incurs $n-1$ memory stalls in the course of $\sigma_{r,1}\cdots\sigma_{r,j_r}$ and the theorem holds. We therefore assume in the following that none of the executions in A's access-stall perturbation sequence is saturated. We will prove that p_n incurs $\Omega(s)$ memory stalls in one of these executions.

For $i \in \{1,\ldots,i_r\}$, let variable n_r^i be defined as follows:

$$n_r^i = \begin{cases} |S_r^m|, & \text{if } \exists m \in \{1,\ldots,j_r\} : B_r^i = O_r^m, \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

Let $N_r = \sum_{i=1}^{i_r} n_r^i$. Thus, it suffices for the proof to show that one of these executions has $N_r = \Omega(s)$. We associate the following integral progress parameter, Φ_r, with each execution $r \ge 0$:

$$\Phi_r = \sum_{i=1}^{i_r} n_r^i \cdot s^{s-i}. \tag{2}$$

If $n_r^i \ge s-1$ for some $0 \le r \le L$ and $i \in \{1,\ldots,i_r\}$, then we are done, since clearly $N_r \ge s-1$ holds in this case. Assume otherwise, then Φ_r can be viewed as an s-digit number in base s whose i'th most significant digit is 0 if $i > i_r$ or equals the number of processes in $S_r^1,\ldots,S_r^{j_r}$ covering B_r^i after α_r otherwise.

From the last property of Definition 4, $O_{r+1}^{j_{r+1}} = B_r^k$, for some $1 \le k \le i_r$ and, moreover, $B_{r+1}^i = B_r^i$ for $i \in \{1,\ldots k\}$, $n_{r+1}^i = n_r^i$ for $i \in \{1,\ldots,k-1\}$, $n_{r+1}^k = n_r^k + 1$, and $n_{r+1}^i = 0$ for $i \in \{k+1,\ldots,i_r\}$. Based on this, we prove that $\Phi_{r+1} > \Phi_r$ (Lemma 13 in the appendix).

Since the sequence $\Phi_1,\ldots\Phi_L$ is strictly growing, each Φ_r is unique. By the definition of Φ, each value Φ_r corresponds to a different partitioning of integer N_r to the values of the s digits of Φ_r. What is the maximum number $\mathcal{N}$ of different executions r for which $N_r \le s$ holds? $\mathcal{N}$ is at most the number of distinguishable partitions of up to s identical balls into s bins. Let $A_{b,c}$ be the number of distinguishable partitions of b identical balls into c bins, then:

$$\mathcal{N} \le \sum_{j=0}^{s} A_{j,s} = A_{s,s+1} = \binom{2s}{s} = \binom{\log L}{\log L/2} = \Theta\left(\frac{4^{\log L/2}}{\sqrt{\pi \log L/2}}\right) = \Theta\left(\frac{L}{\sqrt{\pi \log L/2}}\right) < L.$$

Where the one-before-last equality above follows from Stirling's approximation and the error of the approximation ratio $\binom{\log L}{\log L/2}/\frac{4^{\log L/2}}{\sqrt{\pi \log L/2}}$ is inversely proportional to s [9, page 75]. Thus, for all $L \ge 4$, there is an execution $\alpha_{r'}\sigma_{r',1}\cdots\sigma_{r',j_{r'}}\rho_{r'}$ such that $N_{r'} > s$ holds. □

Together with Lemma 5, we obtain the specific bounds presented in Theorem 14 (in the appendix), and summarized in Table 1.

4 Lower Bound for Randomized Approximate Counters

Proving lower bounds for *randomized* implementations of concurrent objects is more difficult, due to the extra flexibility these implementations have. We were not able to prove general lower bounds for a class of objects, but we take a first step in this direction by proving a lower bound for a specific, but very useful, object, namely an approximate counter. This object allows some error in the operations applied to them. We consider two variants, depending on whether the error is additive or multiplicative.

We assume an *oblivious adversary*, which fixes the sequence of process steps in advance, without being able to predict the coin-flips of the processes or the progress of the execution; in fact, our adversary does not even require knowledge of the implementation, allowing us to prove the lower bound using Yao's Principle [16]. We consider *deterministic algorithms*, since a randomized algorithm can be seen as a weighted average of deterministic ones. A distribution over schedules that gives a high cost on average for any fixed deterministic algorithm, also gives a high cost on average for any randomized algorithm, which also implies that there exists some specific schedule that does so. We will describe an (oblivious) adversary strategy achieving the next lower bound:

Theorem 3 *For any randomized implementation of an m-valued c-multiplicative-accurate counter using historyless primitives for $n \geq m$ processes, and any fixed $\epsilon > 0$, there is an oblivious adversary strategy that yields, with probability at least $1 - \epsilon$, an execution consisting of at most $m - 1$ concurrent* `CounterIncrement` *operation instances, some of which may be incomplete, followed by a* `CounterRead` *operation instance, in which one of the following conditions holds: (a) a constant fraction of the* `CounterIncrement` *instances take more than w operations; (b) the value returned by the* `CounterRead` *operation instance is not consistent with any linearization of the completed operation instances; or (c) the* `CounterRead` *operation instance takes $\Omega\left(\frac{\log\log m - \log\log c}{\log w}\right)$ operations.*

We first consider a schedule constructed as follows. Process p_1 carries out an operation β_1 for at most w steps. With probability p for each step, p_1 is stopped early and is suspended before it can carry the step out; if the step is not a read operation, this means that the target register is now covered by a pending operation that can be delivered later to overwrite any subsequent work by other processes. Whether p_1 completes its operation or not, process p_2 is next scheduled to carry out at most w steps, each of which causes p_2 to be suspended with probability p as before, and this process is repeated for the remaining processes up through p_{n-1}. In this way we assemble a schedule $\Gamma = \beta'_1\beta'_2 \ldots \beta'_{n-1}$, where each β'_i is an initial prefix of some high-level operation β_i.

From this schedule we construct a family of schedules $\{\Xi_k\}_{k\geq 0}$, where each Ξ_k consists of an initial prefix of Γ of length k (i.e., consisting of k steps), followed by the delivery of all delayed operations from Γ, and in turn followed by the first r steps of a single operation α executed by p_n. Thus each Ξ_k is of the form $\beta'_1\beta'_2 \ldots \beta''_m\delta_m\delta_{m-1} \ldots \delta_1\alpha$, where δ_i is either the delayed operation of i or the empty sequence if there is no such operation, α is the single operation of p_n, and β''_m is a prefix of β_m that makes the initial segment have the correct length.

The proof is based first on bounding the number of distinct values returned by the reader across all the schedules Ξ_k as a function of p and r, and then showing that we can select a subset of these schedules that must either violate the restriction to short increment and read operations or return significantly more distinct values. This implies that choosing one of these schedules uniformly at random is likely to hit one of the bad outcomes. The next lemma, whose proof appears in the appendix, bounds the number of distinct return values:

Lemma 4 *Among the schedules Ξ_k above, α returns at most $(1+1/p)^r$ distinct values on average, where the average is taken over the random choices of the adversary for when to delay operations.*

The key idea is that because α is deterministic, the value it returns can depend only on the values of the at most r registers it reads, and that each register will get at most $1 + 1/p$ values on average in all the Ξ_k before it becomes covered by some δ_i. This is essentially the same idea as used in [3] for max registers, except that we provide a more careful analysis of the dependence between the number of values found in each registers, because the union bound used in [3] reduces the lower bound by a $\Theta(\log\log m)$ factor that in our case would eliminate the lower bound completely.

Lemma 4 holds for arbitrary sequences of operations. To prove Theorem 3, we show that for the specific case where $p = 1/4w$ and each β_i is a `CounterIncrement` and α is a `CounterRead` for a c-multiplicative-accurate counter, we can pick out a subfamily of executions $\Xi_{k_0}, \Xi_{k_1}, \ldots \Xi_{k_{\ell-1}}$, where $\ell - 1 = \left\lfloor \frac{1}{2} \log_{2c} \sqrt{m} \right\rfloor - 1 = \Theta(\log m / \log c)$, such that, on average, a constant fraction of the executions Ξ_{k_i} satisfies one of the conditions in Theorem 3. The full proof appears in the appendix.

If we choose w to match the lower bound on `CounterRead`, we get a lower bound on the worst-case cost of any c-multiplicative-accurate counter operation for fixed c of $\Omega\left(\frac{\log\log m}{\log\log\log m}\right)$. This is much smaller than Jayanti's lower bound of $\Omega(\log n)$ on randomized n-bounded counters [14], which also allows much stronger primitives in the implementation. But the smaller bound is not surprising if one considers that a c-multiplicative-accurate counter effectively provides only $\Theta(\log\log m)$ bits of information about the number of increments, compared with $\Theta(\log m)$ for standard counter.

5 Summary

This paper presents lower bounds for concurrent obstruction-free implementations of objects that are used in a restricted manner. (See Table 1 in the introduction.) The step lower-bound on max registers is tight [3] and the step lower bound on randomized counters is almost tight, as there is an $O(\log\log m)$ upper bound [7], under the same adversary model. It is unclear whether the other lower bounds are tight. Another interesting research direction is to devise generic implementations for L-perturbable objects. This is of particular interest in the case of randomized implementations, where there is also an important issue of the type of adversary tolerated.

References

[1] D. Alistarh, J. Aspnes, K. Censor-Hillel, S. Gilbert, and M. Zadimoghaddam. Optimal-time adaptive tight renaming, with applications to counting. In *PODC*, pages 239–248, 2011.

[2] D. Alistarh, J. Aspnes, S. Gilbert, and R. Guerraoui. The complexity of renaming. In *FOCS*, pages 718–727, 2011.

[3] J. Aspnes, H. Attiya, and K. Censor. Polylogarithmic concurrent data structures from monotone circuits. *J. ACM*, to appear. Previous version in *PODC*, pages 36–45, 2009.

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

[5] H. Attiya, R. Guerraoui, D. Hendler, and P. Kuznetsov. The complexity of obstruction-free implementations. *J. ACM*, 56(4), 2009.

[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] C. Dwork, M. Herlihy, and O. Waarts. Contention in shared memory algorithms. *Journal of the ACM*, 44(6):779–805, 1997.

[9] W. Feller. *An Introduction to Probability Theory and Its Applications, Vol. 1.* Wiley, 1968.

[10] F. Fich, M. Herlihy, and N. Shavit. On the space complexity of randomized synchronization. *J. ACM*, 45(5):843–862, 1998.

[11] F. E. Fich, D. Hendler, and N. Shavit. Linear lower bounds on real-world implementations of concurrent objects. In *FOCS*, pages 165–173, 2005.

[12] M. Herlihy, V. Luchangco, and M. Moir. Obstruction-free synchronization: Double-ended queues as an example. In *ICDCS*, pages 522–529, 2003.

[13] M. Herlihy and J. M. Wing. Linearizability: a correctness condition for concurrent objects. *ACM Transactions on Programming Languages and Systems*, 12(3):463–492, June 1990.

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

[15] P. Jayanti, K. Tan, and S. Toueg. Time and space lower bounds for nonblocking implementations. *SIAM Journal on Computing*, 30(2):438–456, 2000.

[16] A. C.-C. Yao. Probabilistic computations: Toward a unified measure of complexity. In *FOCS*, pages 222–227, 1977.

A Examples of L-Perturbable Objects

A *counter* is a linearizable object that supports a `CounterIncrement` operation and a `CounterRead` operation, which returns the number of `CounterIncrement` operation instances linearized before it. In a *k-additive-accurate counter*, any `CounterRead` operation returns a value within $\pm k$ of the number of `CounterIncrement` operation instances linearized before it. A *c-multiplicative-accurate counter* is a counter for which any `CounterRead` operation returns a value x with $v/c \leq x \leq vc$, where v is the number of `CounterIncrement` operation instances linearized before it.

A *max-register* is a linearizable object that supports a `Write` (v) operation, which writes the value v to the object, and a `ReadMax` operation, which returns the maximum value written by any `Write` operation instance linearized before it. In the bounded version of these objects, the object is only required to satisfy its specification if its associated value does not exceed a certain threshold. A *b-bounded max register* takes values in $\{0, \ldots, b-1\}$. A *b-bounded counter* is a counter that takes values in $\{0, \ldots, b-1\}$. For a b-bounded object $\mathcal{O}$, b is the *bound* of $\mathcal{O}$.

Lemma 5 *1. An obstruction-free implementation of a b-bounded-value max register is* $(b-1)$*-perturbable.*

2. An obstruction-free implementation of an m-limited-use max register is $(m-1)$*-perturbable.*

3. *An obstruction-free implementation of an m-limited-use counter is $(\sqrt{m}-1)$-perturbable.*

4. *An obstruction-free implementation of a k-additive-accurate m-limited-use counter is $(\sqrt{\frac{m}{k}}-1)$-perturbable.*

5. *An obstruction-free implementation of an m-limited-use b-valued compare-and-swap object is $(\sqrt[3]{m}-1)$-perturbable (if $b \geq n$).*

6. *An obstruction-free implementation of an m-limited-use collect object is $(m-1)$-perturbable.*

Proof:

1. Let $\mathcal{O}$ be a b-bounded-value max register and consider an obstruction-free implementation of $\mathcal{O}$. We show that $\mathcal{O}$ is $(b-1)$-perturbable for a `ReadMax` operation instance op_n of p_n, by induction, where the base case for $r=0$ is immediate for all objects. We perturb the executions by writing values that increase by one to the max register. This guarantees that op_n has to return different values each time, while getting closer to the limit of the object as slowly as possible.

   Formally, let $r < b$ and let $\alpha_{r-1}\lambda_{r-1}$ be an $(r-1)$-perturbing execution of $\mathcal{O}$. If $\alpha_{r-1}\lambda_{r-1}$ is saturated, then, by case (1) of Definition 2, it is also an r-perturbing execution.

   Otherwise, our induction hypothesis is that op_n returns $r-1$ when run after $\alpha_{r-1}\lambda_{r-1}$. For the induction step, we build an r-perturbing execution after which the value returned by op_n is r. Since $\alpha_{r-1}\lambda_{r-1}$ is not saturated, there is a process $p_\ell \neq p_n$ that does not take steps in λ_{r-1}. Let γ be the execution fragment by p_ℓ where it first finishes any incomplete operation in α and then performs a `Write` operation to the max register with the value $r \leq b-1$. Then op_n returns the value r when run after $\alpha_{r-1}\gamma\lambda_{r-1}$, and $r-1$ when run after the $(r-1)$-perturbing execution $\alpha_{r-1}\lambda_{r-1}$. It follows that r-perturbing executions may be constructed from $\alpha_{r-1}\lambda_{r-1}$ and γ as specified by Definition 2.

2. The proof for an m-limited-use max register is the same as that for a b-bounded value max register. We could even allow writing any increasing sequence of values to the max register rather than only increasing by one, since the limit of the object applies to the number of operations rather than to its value.

3. When $\mathcal{O}$ is an m-limited-use counter, we use a proof similar to the one we used for a limited-use max register, where we perturb a `CounterRead` operation op_n by applying `CounterIncrement` operations. The subtlety in the case of a counter comes from the fact that a single perturbing operation may not be sufficient for guaranteeing that op_n returns a different value after $\alpha_{r-1}\lambda_{r-1}$ and after $\alpha_{r-1}\gamma\lambda_{r-1}$, since we do not know how many of the `CounterIncrement` operations by processes that are active after α_{r-1} were linearized. As there are at most $r-1$ such operations, in order to ensure that different values are returned by p_n after these two executions, we construct γ by letting the process p_ℓ apply r `CounterIncrement` operations after finishing any incomplete operation in α_{r-1}. This can be done as long as $r \leq \sqrt{m}$ in order not to pass the limit on the number of operations allowed, which will be $1+\sum_{r=1}^{\sqrt{m}} r = 1+\frac{(\sqrt{m}-1)\sqrt{m}}{2} \leq m$.

4. For a k-additive-accurate m-limited-use counter the proof is similar to that of a counter, except that p_ℓ needs to perform an even larger number of `CounterIncrement` operations in γ, because of the inaccuracy allowed in the returned value of the `CounterRead` operation op_n. Denote by I_r the number of `CounterIncrement` operation instances performed by the perturbing process in iteration r. We have that $I_1 = k+1$ in order for op_n to return at least 1. We claim that for $r > 1$, $I_r = 2k + r$, and prove this by induction. The operation op_n run after $\alpha_{r-1}\lambda_{r-1}$ can return a value which is as large as $\sum_{j=1}^{r-1} I_j + k$. Therefore, we need the number of complete `CounterIncrement` operation instances after $\alpha_{r-1}\gamma\lambda_{r-1}$ to be at least $\sum_{j=1}^{r-1} I_j + k + (k+1)$, for op_n to return at least $\sum_{j=1}^{r-1} I_j + k + 1$. Besides the `CounterIncrement` operation instances in γ, at least $\sum_{j=1}^{r-1} I_j - (r-1)$ `CounterIncrement` operation instances have finished, therefore setting $I_r = 2k + r$ implies that op_n returns at least $\sum_{j=1}^{r-1} I_j - (r-1) + I_j - k$, which is $\sum_{j=1}^{r-1} I_j + k + 1$ as needed.

   This claim implies that a k-additive-accurate m-limited-use counter is $\left(\sqrt{\frac{m}{k}} - 1\right)$-perturbable, because the total number of operation instances will be

$$
\begin{aligned}
1 + \left((k+1) + (2k+2) + \ldots + \left(2k + \sqrt{\frac{m}{k}} - 1\right)\right) &\leq 1 + 2k\left(\sqrt{\frac{m}{k}} - 1\right) + \frac{\left(\sqrt{\frac{m}{k}} - 1\right)\left(\sqrt{\frac{m}{k}} - 1 + 1\right)}{2} \\
&\leq 1 + \left(2\sqrt{m} - 2k\right) + \frac{m}{2} \\
&\leq m,
\end{aligned}
$$

   where the last inequality holds for a large enough m ($m \geq 16$).

5. Let $\mathcal{O}$ be an m-limited-use b-bounded *compare-and-swap* object, $b \geq n$. We show that it is $(\sqrt[3]{m} - 1)$-perturbable for a *read* operation instance by p_n, by induction, where the base case for $r = 0$ is immediate for all objects. In our construction, all processes except for p_n perform only *CAS* operation instances.

   Let $r < \sqrt[3]{m} - 1$ and let $\alpha_{r-1}\lambda_{r-1}$ be an $(r-1)$-perturbing execution of $\mathcal{O}$. If $\alpha_{r-1}\lambda_{r-1}$ is saturated, then, by case (1) of Definition 2, it is also an r-perturbing execution.

   Otherwise, our induction hypothesis is that $\alpha_{r-1}\lambda_{r-1}$ includes at most $\sum_{i=1}^{r-1} i^2$ *CAS* operation instances. Let u be the value returned by op_n after $\alpha_{r-1}\lambda_{r-1}$, and let j denote the number of processes that apply events in λ_{r-1} and let $p_{r-1}^1, \ldots, p_{r-1}^j$ be these processes. Let ξ be an execution fragment that follows α_{r-1} in which all active processes other than $p_{r-1}^1, \ldots, p_{r-1}^j$ finish any incomplete operation instances they started in α_{r-1}. For $k \in \{1, \ldots, j\}$, let (u_k, v_k) denote the operands of the last *CAS* operation instance started by p_{r-1}^k in α_{r-1}. Since $\alpha_{r-1}\lambda_{r-1}$ is not saturated and since $r - 1 < \sqrt[3]{m} - 1$, there is a process $p_\ell \notin \{p_{r-1}^1, \ldots, p_{r-1}^j\} \cup \{p_n\}$ and, moreover, there is a value $v \in \{1, \ldots, n\} \setminus \{u, u_1, \ldots, u_j\}$.

   Denote by β the sequence of operation instances $CAS(u,v)CAS(v_1,v)\ldots CAS(v_j,v)$, denote by β^r the sequence of operation instances resulting from concatenating r copies of β and let $\gamma = \xi\beta^r$.

   We claim that $\mathcal{O}$'s value after $\alpha_{r-1}\gamma\lambda_{r-1}$ is v. Consider $\mathcal{O}$'s value after p_ℓ executes β once after $\alpha_{r-1}\xi$. There are two possibilities: either $\mathcal{O}$'s value is v (in which case it remains v also after $\alpha\gamma$), or, otherwise, all the *CAS* instances in β failed, implying that one or more of the operation instances performed by $p_{r-1}^1, \ldots, p_{r-1}^j$ are linearized during the execution of γ. In the latter case, consider $\mathcal{O}$'s value after p_ℓ executes β twice after $\alpha_{r-1}\xi$. Once again, either $\mathcal{O}$'s value is v (and remains v also after $\alpha_{r-1}\gamma$), or, otherwise, additional operation instances performed by $p^1_{r-1}, \ldots, p^j_{r-1}$ are linearized during the second execution of β. Applying this argument iteratively and noting that $j \le r-1$, by construction, establishes our claim.

Consider the execution $\alpha_{r-1}\gamma\lambda_{r-1}\phi$, where ϕ is an execution of *read* by p_n. Then ϕ must return v, whereas an execution of *read* by p_n after $\alpha_{r-1}\lambda_{r-1}$ returns $u \neq v$. Execution $\alpha_r\lambda_r$ can now be constructed as in the proofs for limited use max registers and counters. The number of operation instances applied by p_ℓ in γ is $(j+1)\cdot r \le r^2$. Since $\mathcal{O}$ allows only m operation instances, this implies that the sequence can have length $\sqrt[3]{m}-1$, because the total number of operation instances will be $1+\sum_{r=1}^{\sqrt[3]{m}-1} r^2 < m$.

6. Let $\mathcal{O}$ be an m-limited-use collect object and consider an obstruction-free implementation of $\mathcal{O}$. We show that $\mathcal{O}$ is $(m-1)$-perturbable for a *collect* operation instance op_n of p_n, by induction, where the base case for $r=0$ is immediate for all objects. We perturb the executions by having processes store values that change their collect component. This guarantees that op_n has to return different values each time, while getting closer to the limit of the object as slowly as possible.

   Formally, let $r < m$ and let $\alpha_{r-1}\lambda_{r-1}$ be an $(r-1)$-perturbing execution of $\mathcal{O}$. If $\alpha_{r-1}\lambda_{r-1}$ is saturated, then, by case (1) of Definition 2, it is also an r-perturbing execution.

   Otherwise, Let $V =< v_1, \ldots, v_n >$ denote the value that is returned by a *collect* operation by p_n after $\alpha_{r-1}\lambda_{r-1}$. Since $\alpha_{r-1}\lambda_{r-1}$ is not saturated, there is a process $p_\ell \neq p_n$ that does not take steps in λ_{r-1}. Let γ be the execution fragment by p_ℓ where it first finishes any incomplete operation in α and then applies an *update*(v'_ℓ) operation operation to $\mathcal{O}$, for some $v'_\ell \neq v_\ell$. Then op_n must return different values when run after $\alpha_{r-1}\gamma\lambda_{r-1}$ and after the $(r-1)$-perturbing execution $\alpha_{r-1}\lambda_{r-1}$. It follows that r-perturbing executions may be constructed from $\alpha_{r-1}\lambda_{r-1}$ and γ as specified by Definition 2.

□

B Proofs Omitted from Section 3

Lemma 6 *An L-perturbable object implementation from historyless primitives has an access-perturbation sequence of length L.*

Proof: Let $\mathcal{O}$ be an L-perturbable object implementation from historyless primitives. We show that it has an access-perturbation sequence of length L, for the operation op_n as defined in Definition 3. The proof is by induction, where we prove the existence of the execution $\alpha_r\lambda_r\phi_r$, for every r, $0 \le r \le L$. To allow the proof to go through, in addition to proving that the execution $\alpha_r\lambda_r\phi_r$ satisfies the 4 conditions of Definition 3, we will prove that $\alpha_r\lambda_r$ is r-perturbing.

For the base case, $r=0$, $\alpha_0\lambda_0$ is empty and ϕ_0 is an execution of op_n starting from the initial configuration. Moreover, the empty execution is 0-perturbing. We next assume the construction of the sequence up to $r-1 < L$ and construct the next execution $\alpha_r\lambda_r\phi_r$ as follows.

By the induction hypothesis, the execution $\alpha_{r-1}\lambda_{r-1}$ is $(r-1)$-perturbing. If $\alpha_{r-1}\lambda_{r-1}$ is saturated, then, by case (1) of Definition 2, $\alpha_r = \alpha_{r-1}$, $\lambda_r = \lambda_{r-1}$ and $\alpha_r\lambda_r$ is r-perturbing.

Moreover, by property 4(a) of Definition 3, $\alpha_r\lambda_r\phi_r$ is the r'th access-perturbation execution, where $\phi_r = \phi_{r-1}$.

Assume otherwise. Then, by property 4(b) of Definition 3 , there is a process $p_{\ell_r} \neq p_n$ that does not take steps in λ_{r-1}, for which there is an extension of α_{r-1}, γ_r, consisting of events by p_{ℓ_r}, such that p_n returns different responses when performing op_n by itself after $\alpha_{r-1}\lambda_{r-1}$ and after $\alpha_{r-1}\gamma_r\lambda_{r-1}$. As per Definition 2, let $\gamma_r = \gamma_r' e_r \gamma_r''$, where e_r is the first event of γ such that op_n returns different responses after $\alpha_{r-1}\lambda_{r-1}$ and after $\alpha_{r-1}\gamma_r' e_r \lambda_{r-1}$. Clearly e_r is a nontrivial event.

Denote by ϕ_r the execution of op_n by p_n after $\alpha_{r-1}\gamma_r' e_r \lambda_{r-1}$. Since op_n returns different values after $\alpha_{r-1}\lambda_{r-1}$ and after $\alpha_{r-1}\gamma_r' e_r \lambda_{r-1}$, and since the implementation uses only historyless primitives, this implies that e_r is applied to some base object B not in $\{O^1_{r-1}, \ldots, O^{j_r-1}_{r-1}\}$ that is accessed by p_n in ϕ_r.

We define λ'_{r-1} to be the subsequence of λ_{r-1} containing all events to base objects that are not accessed by p_n in ϕ_r, and λ''_{r-1} to be the subsequence of λ_{r-1} containing all events to base objects that are accessed by p_n in ϕ_r. We then define $\alpha_r = \alpha_{r-1}\gamma_r'\lambda'_{r-1}$, $\lambda_r = \lambda''_{r-1}e_r$ and show that $\alpha_r\lambda_r\phi_r$ satisfies the properties of Definition 3.

We first observe that $\alpha_r\lambda_r\phi_r$ is a well defined execution, since the execution fragment γ_r' by p_{ℓ_r} is performed after α_{r-1}, and all operations in λ_{r-1} are nontrivial events to distinct base objects none of which is by p_{ℓ_r}. It follows that $\alpha_r\lambda_r$ and $\alpha_{r-1}\gamma_r' e_r\lambda_{r-1}$ are indistinguishable to p_n, hence ϕ_r is a solo execution of op_n by p_n after both executions.

Property 1 holds since $\alpha_r\lambda_r$ is p_n-free by construction, and ϕ_r is a solo execution fragment by p_n in which it performs op_n, so Property 2 holds. To show Property 3, we observe that $\alpha_r\lambda_r$ is indistinguishable to p_n from $\alpha_{r-1}\gamma_r' e_r\lambda_{r-1}$ and hence p_n accesses the base object B in ϕ_r. Finally, Property 4 follows by construction.

We conclude the proof by claiming that $\alpha_r\lambda_r$ is r-perturbing, which follows from its construction and Definition 2. □

Lemma 7 $\Psi_r > \Psi_{r-1}$.

Proof:

$$\begin{aligned}
\Psi_r &= \textstyle\sum_{j=1}^{i_r} b_r^j \cdot \frac{L}{2^j} \\
&= \textstyle\sum_{j=1}^{k-1} b_r^j \cdot \frac{L}{2^j} + b_r^k \cdot \frac{L}{2^k} + \sum_{j=k+1}^{i_r} b_r^j \cdot \frac{L}{2^j} \\
&= \textstyle\sum_{j=1}^{k-1} b_{r-1}^j \cdot \frac{L}{2^j} + \frac{L}{2^k} + \sum_{j=k+1}^{i_r} b_r^j \cdot \frac{L}{2^j} \\
&\geq \textstyle\sum_{j=1}^{k-1} b_{r-1}^j \cdot \frac{L}{2^j} + \frac{L}{2^k} \\
&> \textstyle\sum_{j=1}^{k-1} b_{r-1}^j \cdot \frac{L}{2^j} + \sum_{j=k+1}^{i_{r-1}} b_{r-1}^j \frac{L}{2^j} \\
&= \Psi_{r-1},
\end{aligned}$$

where the last equality is based on the observation that $b_{r-1}^k = 0$. □

Theorem 8 *An n-process obstruction-free implementation of an m-limited-use max register, m-limited-use counter, m-limited-use b-valued compare-and-swap object or an m-limited-use collect object from historyless primitives has an operation instance requiring $\Omega(\min(\log m, n))$ steps. An obstruction-free implementation of a b-bounded max register from historyless primitives has an operation instance requiring $\Omega(\min(\log b, n))$ steps. An obstruction-free implementation of a k-additive-accurate m-limited-use counter from historyless primitives has an operation instance requiring $\Omega(\min(\log m - \log k, n))$ steps.*

Definition 5 *A* cover-perturbation sequence of length $1 \leq L \leq n-1$ *of an operation instance* op_n *by process* p_n *on an object* $\mathcal{O}$ *is a sequence of executions* $\{\alpha_r\lambda_r\phi_r\}_{r=0}^{L}$*, such that* $\alpha_0\lambda_0$ *is empty,* ϕ_0 *is an execution of* op_n *by* p_n*, and for every* r*,* $1 \leq r \leq L$*, the following hold.*

1. *The execution* $\alpha_r\lambda_r$ *is* p_n*-free.*
2. *In* ϕ_r*, process* p_n *runs solo after* $\alpha_r\lambda_r$ *until it completes* op_n*.*
3. *In* λ_r*, distinct processes* $q_1, \ldots, q_r$ *each apply a nontrivial event to distinct base objects* $O_1, \ldots, O_r$*, respectively.*
4. $|active(\alpha_r\lambda_r)| \leq r$*.*

The space lower bound for cover-perturbable objects follows immediately from Property 3:

Theorem 9 *Let* A *be an* n*-process obstruction-free implementation of an object* $\mathcal{O}$ *from historyless primitives. If* A *has a cover-perturbation sequence of length* L*, then* A *has an execution in which* L *distinct base objects are accessed.*

The next lemma shows that every L-perturbable object has a cover-perturbation sequence of length L.

Lemma 10 *An* L*-perturbable object has a cover-perturbation sequence of length* L*.*

Proof: We show that the object has a cover-perturbation sequence of length L for operation instance op_n as defined in Definition 5. The proof is by induction, where we prove the existence of the execution $\alpha_r\lambda_r\phi_r$, for every r, $0 \leq r \leq L$. To allow the proof to go through, in addition to proving that the execution $\alpha_r\lambda_r\phi_r$ satisfies the four conditions of Definition 5, we will prove that $\alpha_r\lambda_r$ is r-perturbing.

For the base case, $r = 0$, $\alpha_0\lambda_0$ is empty and ϕ_0 is an execution of op_n starting from the initial configuration. Moreover, the empty execution is 0-perturbing. We next assume the construction of the sequence up to $r - 1 < L$ and construct the next execution $\alpha_r\lambda_r\phi_r$ as follows.

By the induction hypothesis, the execution $\alpha_{r-1}\lambda_{r-1}$ is $(r-1)$-perturbing. If α_{r-1} is saturated, we take $\alpha_r = \alpha_{r-1}$ and $\lambda_r = \lambda_{r-1}$. Otherwise, by case (2) of Definition 2, there is a process $p_{\ell_r} \neq p_n$ that does not take steps in λ_{r-1}, for which there is an extension of α_{r-1}, γ_r, consisting of events by p_{ℓ_r}, such that p_n returns different responses when performing op_n by itself after $\alpha_{r-1}\lambda_{r-1}$ and after $\alpha_{r-1}\gamma_r\lambda_{r-1}$. As per Definition 2, let $\gamma_r = \gamma'_r e_r \gamma''_r$, where e_r is the first event of γ such that op_n returns different responses after $\alpha_{r-1}\lambda_{r-1}$ and after $\alpha_{r-1}\gamma'_r e_r\lambda_{r-1}$. Clearly e_r is a nontrivial event.

Denote by ϕ_r the execution of op_n by p_n after $\alpha_{r-1}\gamma'_r e_r\lambda_{r-1}$. Since op_n returns different values after $\alpha_{r-1}\lambda_{r-1}$ and after $\alpha_{r-1}\gamma'_r e_r\lambda_{r-1}$, and since the implementation uses only historyless primitives, this implies that e_r is applied to some base object B not in $\{O_1, \ldots, O_{r-1}\}$ that is accesses by p_n in ϕ_r.

Define $\alpha_r = \alpha_{r-1}\gamma'$ and $\lambda_r = \lambda_{r-1}e$. To conclude the proof, we need to show that the execution $\alpha_r\lambda_r$ satisfies the properties of Definition 5. Since Property 1 holds for execution $\alpha_{r-1}\lambda_{r-1}$, it is p_n-free. By construction, γ' is performed by $p_{\ell_r} \neq p_n$, hence $\alpha_r\lambda_r$ is also p_n-free, establishing that Property 1 holds for it as well. Property 4 holds for $\alpha_r\lambda_r$ since $|active(\alpha_r\lambda_r)| = |active(\alpha_{r-1}\lambda_{r-1})| + 1 \leq r - 1 + 1 = r$. Finally, Properties 3 and 2 are immediate from our construction.

By its construction, $\alpha_r\lambda_r$ is r-perturbing, which concludes the proof. □

Theorem 11 *The space complexity of any obstruction-free implementation of an m-limited-use max register or an m-limited-use collect object from historyless primitives is $\Omega(\min(m,n))$. The space complexity of any obstruction-free implementation of an m-limited-use b-valued compare-and-swap object from historyless primitives is $\Omega(\min(\sqrt[3]{m},n))$. The space complexity of any obstruction-free implementation of a b-bounded max register from historyless primitives is $\Omega(\min(b,n))$. The space complexity of any obstruction-free implementation of a k-additive-accurate m-limited-use counter from historyless primitives is $\Omega(\min(\sqrt{\frac{m}{k}},n))$.*

Lemma 12 *An L-perturbable object implementation has an access-stall perturbation sequence of length L.*

Proof: Let $\mathcal{O}$ be an L-perturbable object implementation. We show that it has an access-stall-perturbation sequence of length L, for the operation op_n as specified in Definition 4. The proof is by induction, where we prove the existence of the execution $\alpha_r\sigma_{r,1}\cdots\sigma_{r,j_r}\rho_r$, for every r, $0 \le r \le L$.

For the base case, $r = 0$, α_0 is empty, and $j_0 = 0$, implying that λ_0 is also empty. It follows that $\alpha_0\lambda_0$ is the empty execution and therefore, by Definition 2, is 0-perturbing. We next assume the construction of the sequence up to $r < L$ and construct the next access-stall execution, $\alpha_{r+1}\sigma_{r+1,1}\cdots\sigma_{r+1,j_{r+1}}\rho_{r+1}$.

By induction hypothesis, $\alpha_r\sigma_{r,1}\cdots\sigma_{r,j_r}\rho_r$ is an r-perturbing execution. If it is saturated, then we set $\alpha_{r+1} = \alpha_r$, $j_{r+1} = j_r$, $\sigma_{r+1,j} = \sigma_{r,j}$ for $j = 1,\ldots,j_r$ and $\rho_{r+1} = \rho_r$. By induction hypothesis and Definitions 2 and 4, $\alpha_{r+1}\sigma_{r+1,1}\cdots\sigma_{r+1,j_{r+1}}\rho_{r+1}$ is an $r+1$ access-stall execution.

Assume, then, that $\alpha_r\sigma_{r,1}\cdots\sigma_{r,j_r}\rho_r$ is not saturated. Let ϕ_r denote a solo execution of op_n by p_n after $\alpha_r\lambda_r$. Since all the events in λ_r are by distinct processes other than p_n, and since each of the objects O_r^j is accessed by p_n after it is accessed by the processes of S_r^j, for $j \in \{1,\ldots j_r\}$, executions $\alpha_r\sigma_{r,1}\cdots\sigma_{r,j_r}\rho_r$ and $\alpha_r\lambda_r\phi_r$ are indistinguishable to all processes. Since $\alpha_r\lambda_r$ is an r-perturbing execution and $r < L$, and since $\alpha_r\lambda_r$ is not saturated, it follows from Definition 2 that there exists a process $p_{\ell_{r+1}} \neq p_n$ that applies no event in λ_r and an extension $\gamma'_{r+1}e_{r+1}$ of α_r, consisting of events by $p_{\ell_{r+1}}$, such that e_{r+1} is the first event of $\gamma'_{r+1}e_{r+1}$ such that p_n returns different responses after $\alpha_r\lambda_r$ and after $\alpha_r\gamma'_{r+1}e_{r+1}\lambda_r$. It follows that e_{r+1} is a nontrivial event applied by $p_{\ell_{r+1}}$ to a base object in $\{B_r^1,\ldots,B_r^{i_r}\}$; let this base object be B_r^k. There are two cases:

Case 1: If $B_r^k = O_r^{k'}$, for some $k' \in \{1,\ldots,j_r\}$, then let $j_{r+1} = k'$, for $j = 1,\ldots,k'-1$, $\sigma_{r+1,j} = \sigma_{r,j}$ (thus, $O_{r+1}^j = O_r^j$ and $S_{r+1}^j = S_r^j$), $O_{r+1}^{k'} = O_r^{k'}$, $S_{r+1}^{k'} = S_r^{k'} \cup \{p_{\ell_{r+1}}\}$ (thus, e_{r+1} appears in $\sigma_{r+1,k'}$), $\alpha_{r+1} = \alpha_r\gamma'_{r+1}\lambda'_r$ and $\lambda_{r+1} = \lambda''_r$, where λ'_r consists of the events of λ_r applied to objects $O_r^{k'+1},\ldots,O_r^{j_r}$, and λ''_r consists of the events of λ_r applied to objects $O_r^1,\ldots,O_r^{k'}$ and the event e_{r+1}.

Case 2: Otherwise, let k' be the largest integer such that $O_r^{k'}$ precedes B_r^k in π_r (or 0 if B_r^k is not preceded in π_r by any of the objects $O_r^1,\ldots,O_r^{j_r}$). Then $j_{r+1} = k'+1$, for $j = 1,\ldots,k'$, $\sigma_{r+1,j} = \sigma_{r,j}$ (hence also $O_{r+1}^j = O_r^j$ and $S_{r+1}^j = S_r^j$), $O_{r+1}^{j_{r+1}} = B_r^k$, $S_{r+1}^{j_{r+1}} = \{p_{\ell_{r+1}}\}$, $\alpha_{r+1} = \alpha_r\gamma'_{r+1}\lambda'_r$ and $\lambda_{r+1} = \lambda''_r$, where λ'_r consists of the events of λ_r applied to objects $O_r^{k'+1},\ldots,O_r^{j_r}$, and λ''_r consists of the events of λ_r applied to objects $O_r^1,\ldots,O_r^{k'}$ and the event e_{r+1}, and in $\sigma_{r+1,j_{r+1}}$, p_n applies events until it is about to apply its first event to $O_{r+1}^{j_{r+1}}$, then $p_{\ell_{r+1}}$ applies e_{r+1} and finally p_n applies its first event to $O_{r+1}^{j_{r+1}}$.

In both cases, it follows from the construction and from Definition 2 that $\alpha_{r+1}\lambda_{r+1}$ is (r+1)-perturbing. Since α_r is p_n-free and none of the events of $\gamma'_{r+1}\lambda'_r$ are by p_n, α_{r+1} is also p_n-free. Let ρ_{r+1} denote the execution in which process p_n runs solo after $\alpha_{r+1}\sigma_{r,1}\cdots\sigma_{r,j_{r+1}}$ until it completes the operation instance op_n, in the course of which it accesses the base objects $B^1_{r+1}, \ldots, B^{i_{r+1}}_r$. It follows from our construction that $\alpha_{r+1}\sigma_{r+1,1}\cdots\sigma_{r+1,j_{r+1}}\rho_{r+1}$ is an $r+1$ access-stall execution. □

Lemma 13 $\Phi_{r+1} > \Phi_r$.

Proof:

$$\begin{array}{rcl}
\Phi_{r+1} & = & \sum_{i=1}^{i_{r+1}} n^i_{r+1} \cdot s^{s-i} \\
& = & \sum_{i=1}^{k} n^i_{r+1} \cdot s^{s-i} \\
& = & \sum_{i=1}^{k-1} n^i_r \cdot s^{s-i} + (n^k_r + 1) \cdot s^{s-k} \\
& > & \sum_{i=1}^{k} n^i_r \cdot s^{s-i} + \sum_{i=k+1}^{s} (s-1) \cdot s^{s-i} \\
& \geq & \sum_{i=1}^{i_r} n^i_r \cdot s^{s-i} \\
& = & \Phi_r
\end{array}$$

□

Theorem 14 *An n-process obstruction-free implementation of an m-limited-use max register, m-limited-use counter, an m-limited-use b-valued compare-and-swap object or an an m-limited-use collect object from any read-modify-write primitives has an operation instance that either requires $\Omega(\min(\log m, n))$ steps or incurs $\Omega(\min(\log m, n))$ stalls. An obstruction-free implementation of a b-bounded max register from any read-modify-write primitives has an operation instance that either requires $\Omega(\min\big(\log b, n\big))$ steps or incurs $\Omega(\min\big(\log b, n\big))$ stalls. An obstruction-free implementation of a k-additive-accurate m-limited-use counter from any read-modify-write primitives has an operation instance that either requires $\Omega(\min\big(\log m - \log k, n\big))$ steps or incurs $\Omega(\min\big(\log m - \log k, n\big))$ stalls.*

C Proofs Omitted from Section 4

Lemma 4 (repeated) *Among the schedules Ξ_k defined in Section 4, α returns at most $(1+1/p)^r$ distinct values on average, where the average is taken over the random choices of the adversary for when to delay operations.*

Proof: For simplicity, let us assume that α performs only read operations; as observed in [3], for the particular class of executions we are considering we can always replace an α that does not perform only read operations by an optimized version that reads each register on its first access and replaces any subsequent accesses with internal simulations. Notice that this means we allow the algorithm to be aware of our set of restricted executions.

For each Ξ_k, define a bit-vector $b^k = b^k_1, b^k_2, \ldots b^k_r$, where each b^k_i is 1 if either there are fewer than i reads in α or the i-th read observes a value that was written by some δ_j, and 0 if the i-th read in α reads a register that was not written by some delayed operation δ_j.

We will compute an upper bound $T(b^k)$ on the expected number of distinct sequences of values that α may observe in $\Xi_k, \Xi_{k+1}, \ldots$ as a function of the bit-vector b^k. The expectation is taken over the sequence of adversary choices used to construct the parent schedule Γ, conditioned on the choices, made in the first k steps of the construction, that contribute to Ξ_k. Note that the count includes the sequence of values observed by α in Ξ_k, so, for example, $T(111111)$ will be 1 (since no further changes in the bit-vector are possible) and not 0.

Moving from Ξ_k to Ξ_{k+1} involves adding one step, which may change at most one register. To be visible to α, this register must not be covered in Ξ_k and must appear somewhere in the sequence of registers that α reads. The effect of changing the i-th register that α reads is that (a) α sees a new sequence of values, and (b) α can change what registers it reads on steps $i+1, i+2, \ldots, r$. It may also be the case (with probability p, independent of previous choices) that b_i^{k+1} becomes 1, because the new operation on the i-th register becomes one of the covering operations δ_j.

From the point of view of the b^k vector, the effect of adding an operation that changes the sequence of values seen by α is to (a) set b_i^{k+1} to 1 with probability p, for some i with $b_i^k = 0$; and (b) allow any later bits in b^{k+1} to be set arbitrarily. We can bound the number of distinct sequences seen by α by bounding the number of times this can happen before all bits in b^k are 1.

Define the following recurrence on suffixes of b. Here $T(x)$ represents an upper bound on the expected number of distinct sequences of values that can be obtained from the last $|x|$ registers read by α, if we do not change the value of any earlier registers.

$$\begin{aligned}
T(\langle\rangle) &= 1. \\
T(1x) &= T(x). \\
T(0x) &= T(x) + (1-p)\max_{x'} T(0x') + p \max_{x'} T(1x').
\end{aligned}$$

The justification for this recurrence is that (a) there is only one possible sequence of values that can be obtained by reading no registers; (b) if the first register in the suffix is covered, then any further values must be obtained by updating later registers; and (c) if the first register is not covered, then we obtain at most $T(x')$ distinct values in the tail before having to apply an operation that changes the value in the first register, after which we get $T(0x')$ more values if we are lucky enough not to cover it and $T(1x')$ more values otherwise, where x' is the new tail that holds after the first register is changed.

We can eliminate the max functions in the last line by observing that setting x' to the all-zero vector maximizes $T(1x')$ and $T(0x')$. This is because any sequences of values that can be obtained starting with some registers covered can be obtained using the same strategy when those registers are not covered but the adversary chooses not to use them. So we can rewrite the last line as

$$T(0x) = T(x) + (1-p)T(00^{|x|}) + pT(10^{|x|}),$$

where $|x|$ is the length of x, so that $0^{|x|}$ is the all-zero vector of the same length.

For the vector 0^ℓ, this gives the recurrence

$$T(0^\ell) = T(0^{\ell-1}) + (1-p)T(0^\ell) + pT(10^{\ell-1}) = T(0^{\ell-1}) + (1-p)T(0^\ell) + pT(0^{\ell-1}),$$

which we can solve to get

$$T(0^\ell) = \frac{1+p}{p} T(0^{\ell-1}).$$

It follows that $T(0^\ell) = (1+1/p)^\ell$. Setting $\ell = r$ completes the proof. □

Theorem 3 (repeated) *For any randomized implementation of an m-valued c-multiplicative-accurate counter for $n \geq m$ processes, and any fixed $\epsilon > 0$, there is an oblivious adversary strategy that yields, with probability at least $1-\epsilon$, an execution consisting of at most $m-1$ concurrent* `CounterIncrement` *operation instances, some of which may be incomplete, followed by a*

`CounterRead` operation instance, in which one of the following conditions holds: (a) a constant fraction of the `CounterIncrement` instances take more than w operations; (b) the value returned by the `CounterRead` operation instance is not consistent with any linearization of the completed operation instances; or (c) the `CounterRead` operation instance takes $\Omega\left(\frac{\log\log m - \log\log c}{\log w}\right)$ operations.

Proof: Let $p = 1/4w$, let each β_i be a `CounterIncrement` and let α be a `CounterRead` for a c-multiplicative-accurate counter. We pick out a subfamily of executions $\Xi_{k_0}, \Xi_{k_1}, \ldots \Xi_{k_{\ell-1}}$ where we choose k_i so that the schedule Ξ_{k_i} includes the first $m_i = \left\lfloor (2c)^{2i}\sqrt{m} \right\rfloor$ calls to `CounterIncrement`, where i ranges from 0 to $\ell - 1 = \left\lfloor \frac{1}{2}\log_{2c}\sqrt{m} \right\rfloor - 1 = \Theta(\log m/\log c)$. Call an execution Ξ_{k_i} *good* if (a) it includes at most $m_i/4$ `CounterIncrement` operations that fail to terminate after executing w operations and (b) the `CounterRead` operation returns a value after at most r operations that is consistent with some linearization of the execution.

Our choice of $p = 1/4w$ implies that any `CounterIncrement` is truncated with probability at most 1/4. Standard Chernoff bounds then give that the number of truncated increments among the first m_i increments is at most $m_i/4 + O(\sqrt{m_i \log m_i}) = m_i(1/4 + o(1))$ with probability at least $1 - m^{-a}$ for any fixed constant a (this uses the fact that $m_i = \Omega(\sqrt{m})$); this implies that the same bound holds for all $\ell + 1 \ll n$ executions with probability at least $1 - m^{-a+1}$. Adding these truncated increments to the at most $m_i/4$ increments that fail to finish gives at most $m_i(\frac{1}{2} + o(1)) < \frac{3}{4}m_i$ partial increments in each good Ξ_{k_i} with high probabilitiy for sufficiently large m.

It follows that if v_i is the number of increments that complete before the `CounterRead` in a good execution Ξ_{k_i}, we have $m_i/4 < v_i \leq m_i$ and for the return value x_i of a correct c-multiplicative-accurate `CounterRead` we have $m_i/4c < x_i \leq cm_i$. From our definition of m_i, we have $m_i = m_{i+1}/4c^2$, so $x_i \leq cm_i = cm_{i+1}/4c^2 = m_i/4c < x_{i+1}$; it follows that the return value of the `CounterRead` in any two good executions is distinct.

From Lemma 4, across all executions Ξ_k (including our chosen executions Ξ_{k_i}, there are an expected $(4w+1)^r$ possible distinct return values for `CounterRead` operations that finish in less than r steps. Markov's inequality gives a probability of at least 1/2 that the actual number of return values is bounded by $2(4w+1)^r$. So with probability at least $1/2 - o(1)$, there are at most $2(4w+1)^r$ good executions.

We now have the adversary choose one of the ℓ executions Ξ_{k_i} uniformly at random. Taking into account the probability of errors so far, the probability that the adversary chooses a good execution is at most $(1/2 - o(1))2(4w+1)^r/\ell = (1 - o(1))(4w+1)^r/\ell$. Let $(1 - o(1))(4w+1)^r/\ell = \epsilon$, and solve for r by taking the log of both sides:

$$o(1) + r\log(4w+1) - \log\ell \geq \log\epsilon$$

which gives

$$r \geq \frac{\log\ell - \log(1/\epsilon) - o(1)}{\log(4w+1)} = \Omega\left(\frac{\log\log m - \log\log c}{\log w}\right),$$

where the $\log(1/\epsilon)$ and $o(1)$ terms disappear into the constant. If r is smaller than this bound, then the proportion of good executions will drop to less than ϵ.

So far we have been considering only deterministic counter implementations. However, any randomized counter implementation can be treated as a random superposition of deterministic counter implementations, each characterized by some fixed set of coins, and our adversary strategy does not depend on the particular deterministic implementation chosen. By Yao's Principle, the same probability of getting a bad execution applies even for randomized algorithms. $\square$