Lower Bounds for Restricted-Use Objects*

James Aspnes† Keren Censor-Hillel‡ Hagit Attiya§ Danny Hendler¶

March 13, 2016

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 by proving lower bounds on their space complexity and on the time 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 n-process implementations that use only historyless primitives, we prove $\Omega(\min(L, n))$ space complexity lower bounds, which hold for both deterministic and randomized implementations. For deterministic implementations, we prove lower bounds of $\Omega(\min(\log L, n))$ on the worst-case step complexity of an operation. 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 our deterministic time lower bounds, the paper establishes lower bounds on the expected step complexity of restricted-use randomized versions of many of these objects in a weak oblivious adversary model.

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

*A preliminary version of this paper appeared in [5].

†Department of Computer Science, Yale University. Supported in part by NSF grant CCF-0916389.

‡Department of Computer Science, Technion. Shalon Fellow. Part of this work was done while the author was at the Computer Science and Artificial Intelligence Laboratory, MIT, and supported in part by the Simons Postdoctoral Fellows Program.

§Department of Computer Science, Technion. Supported in part by the *Israel Science Foundation* (grants number 1227/10, 1749/14).

¶Department of Computer Science, Ben-Gurion University. Supported in part by the *Israel Science Foundation* (grants number 1227/10, 1749/14).

Toueg [17], 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.* [17] 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, at least some restricted-use perturbable objects admit implementations that "beat" the lower bound of [17]. 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 [3]. Such a limited-use counter leads to a randomized consensus algorithm with $O(n)$ individual step complexity [6], while limited-use counters and bounded-value max registers are used in a mutual exclusion algorithm with sub-logarithmic amortized work [9]. The max register was also generalized into a two-component max register, in which components are updated in a coordinated manner; this object was used to construct an atomic snapshot object with $O(\log^2 b \log n)$ step complexity for update operations and $O(\log b)$ step complexity for scan operations, where b is the number of updates [4].

This raises the natural question of determining lower bounds on the complexity of restricted-use objects. The proof of Jayanti *et al.* [17] breaks for restricted-use objects because the executions constructed by this proof 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, 8] and generic perturbable objects [16]. 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 n-process L-perturbable objects, we show $\Omega(\min(L, n))$ space complexity lower bounds on obstruction-free implementations from historyless primitives. These lower bounds hold for both deterministic and randomized implementations. For deterministic implementations of these objects, we show $\Omega(\min(\log L, n))$ step complexity lower bounds, using a proof technique that we call *backtracking covering*, introduced by Fich, Hendler and Shavit in [13] and later used in [7].

We also consider deterministic implementations that can apply *arbitrary* primitives and not only

[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 ($m = \text{poly}(n)$)
compare & swap	$m/2 - 1$	$\Omega(\min(\log m, n))$	$\Omega(\min(\log m, n))$	$\Omega(\min(m, n))$	$\Omega\left(\frac{\log n}{\log\log n}\right)$
collect	$m - 1$	$\Omega(\min(\log m, n))$	$\Omega(\min(\log m, n))$	$\Omega(\min(m, n))$	$\Omega\left(\frac{\log n}{\log\log n}\right)$
max register	$m - 1$	$\Omega(\min(\log m, n))$ (also [3])	$\Omega(\min(\log m, n))$	$\Omega(\min(m, n))$	$\Omega\left(\frac{\log n}{\log\log n}\right)$ (for $m \leq n$, also [3])
counter	$\sqrt{m} - 1$	$\Omega(\min(\log m, n))$ (also [3])	$\Omega(\min(\log m, n))$	$\Omega(\min(\sqrt{m}, n))$	$\Omega\left(\frac{\log\log n}{\log\log\log n}\right)$
k-additive counter	$\sqrt{\frac{m}{k}} - 1$	$\Omega(\min(\log m - \log k, n))$ (also [3])	$\Omega(\min(\log m - \log k, n))$	$\Omega(\min(\sqrt{\frac{m}{k}}, n))$	$\Omega\left(\frac{\log\log n}{\log\log\log n}\right)$ (for $k \in O(1)$)

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.

historyless primitives, and use the *memory stalls* measure [10] 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. Table 1 summarizes the lower bounds for specific L-perturbable objects.

We also investigate the expected step complexity of randomized implementations of these objects. We establish a lower bound of $\Omega(\log\log n/\log\log\log n)$ on the expected step complexity of several classes of approximate counters, as well as $\Omega(\log n/\log\log n)$ lower bounds on several stronger objects. Our randomized lower bound technique employs Yao's Principle [19] and assumes a weak oblivious adversary.

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 implementations and an $\Omega(\log m/\log\log m)$ 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 do 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*, specified by the process applying the event, the type of primitive operation applied, and a (possibly empty) list of event operands. A *step* taken by a process consists of local computation followed by one shared memory event.

A *configuration* specifies the state of the system, that is, the internal states of all processes and the values of all base objects. An event is *pending* in configuration C if it is about to be applied in configuration C. An event may return different responses in different configurations, as a function of the value of the base object to which it is applied in the configuration in which it is applied. 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*. *An event e is nontrivial in configuration C* if e is pending in C and will change the value of the base object to which it is applied if performed in C. It follows that e is nontrivial only if it is an application of a nontrivial operation. Note that an application of a nontrivial operation may result in a trivial event, if it is about to write to a base object b a value equal to b's current value.

Let o be a base object that is accessed with two primitives f and f'; f *overwrites* f' on o [12], 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 (register-to-memory) swap primitives is an example of a historyless set of primitives. An object is historyless if it is accessed only by a set of historyless 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). For execution fragments α and β, we let $\alpha\beta$ denote the execution fragment which results when the events of β are concatenated to those of α.

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).

An execution is *well-defined* if it may result when processes each perform a sequence of operation instances according to their algorithms. All the executions we consider are well-defined.

An operation instance is *complete* in an execution if it starts and terminates during the execution, and *incomplete*, if it starts during the execution but does not terminate. Operation instances which are not interleaved are called *non-overlapping*. The *sequential specification* of a data structure restricts its behavior only when all operations instances are non-overlapping. The semantic requirements for all possible concurrent executions are enforced by requiring them to be *linearizable* [15]. This means that for any execution, there is a sequence that contains all the completed operation instances, as well as some of the incomplete ones, that

1. extends the order of non-overlapping operations; and
2. preserves the sequential specification of the object.

An implementation is *obstruction-free* [14] if a process terminates its operation instance if it runs in isolation long enough.

A process p is *active* after an 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 is about to apply a nontrivial operation to 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 bounded-value objects, whose state is associated with a value that cannot exceed some bound. Examples are bounded max-registers and bounded counters [3], whose definitions we now provide. A *counter* is a linearizable object that 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 before it. In the sequential specification of a *k-additive-accurate counter*, a `CounterRead` operation instance returns a value within $\pm k$ of the number of `CounterIncrement` operation instances before it. Similarly, in the sequential specification of a *c-multiplicative-accurate counter*, a `CounterRead` operation instance returns a value x with $v/c \leq x \leq vc$, where v is the number of `CounterIncrement` operation instances before it. The *activity counter* of [9] is closely related to a c-multiplicative-accurate n-bounded-value counter. The difference is that, for the activity counter, the accuracy guarantee holds only with high probability and does not apply for counter values less than $O(\log^4 n)$.

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; in its sequential specification, `ReadMax` returns the maximum value written by a `Write` operation instance 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 $\mathcal{O}$.

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 $u, v \in \{1, \ldots, b\}$. In the sequential specification of the compare-and-swap object, if 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 Bounded Perturbable Objects

Our starting point is the definition of *perturbable* objects by Jayanti *et al.* [17]. 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.

Definition 1 *(See Figure 1.) An obstruction-free implementation of 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*

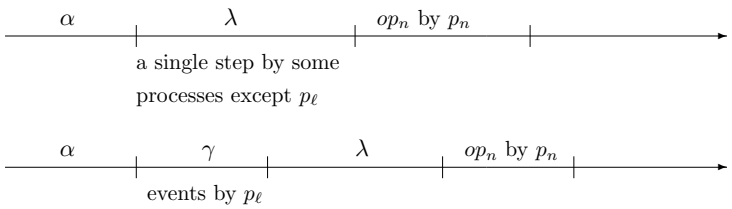

Figure 1: A perturbable object: op_n returns different responses in the two executions.

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 $\alpha\gamma\lambda$.

An object $\mathcal{O}$ is *perturbable* if all its obstruction-free implementations are perturbable.

We observe that $\alpha\gamma\lambda$ in the above definition is a well-defined execution if $\alpha\lambda$ is well-defined. This is because no process applies more than a single event in λ and p_ℓ applies no events in λ, hence no process can distinguish between the two executions before it applies its last event.[2]

The linear lower bounds [17] on the space and step complexity of obstruction-free implementations of 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 *(See Figure 2.) Let $\mathcal{I}$ be an obstruction-free implementation of an object. We say that $\mathcal{I}$ is L-perturbable (for $L \geq 0$) if there is an operation instance op_n by process p_n such that a non-empty set of executions of $\mathcal{I}$, denoted by S_L, can be inductively constructed as follows.*

1. *S_0 is the singleton set containing the empty sequence; otherwise, assume $0 < k \leq L$.*
2. *If $\alpha_{k-1}\lambda_{k-1}$ is in S_{k-1}, where λ_{k-1} consists of $n-1$ events, one by each of the processes $p_1, \ldots, p_{n-1}$, then $\alpha_{k-1}\lambda_{k-1}$ is in S_k. In this case, we say that $\alpha_{k-1}\lambda_{k-1}$ is* saturated*.*
3. *If $\alpha_{k-1}\lambda_{k-1}$ is in S_{k-1} where λ_{k-1} consists of $n-2$ or less events, each by a distinct process, then there is a sequence γ of events by a process p_l different from p_n and the processes that have events in λ_{k-1}, such that the sequences of events by p_n as it performs op_n after $\alpha_{k-1}\lambda_{k-1}$ and $\alpha_{k-1}\gamma\lambda_{k-1}$ differ. Let $\gamma = \gamma' e \gamma''$, where e is the first event of γ such that the sequences of events taken by p_n as it performs op_n by itself after $\alpha_{k-1}\lambda_{k-1}$ and after $\alpha_{k-1}\gamma' e \lambda_{k-1}$ differ. Let λ be some permutation of the event e together with the events in λ_{k-1}, and let λ', λ'' be*

[2]It is, however, possible that an event e in λ returns different responses in executions $\alpha\lambda$ and $\alpha\gamma\lambda$, since e may be applied in different configurations in these two executions.

any two sequences of events such that $\lambda = \lambda'\lambda''$. *Then the execution* $\alpha_k\lambda_k$ *is in* S_k, *where* $\alpha_k = \alpha_{k-1}\gamma'\lambda'$ *and* $\lambda_k = \lambda''$.

For $k \in \{0, \ldots, L\}$,*we call* S_k *the set of k-perturbing executions with respect to* op_n.

An object is *L-perturbable* if all its obstruction-free implementations are L-perturbable. 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 $0 \leq k \leq L$. 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 $\alpha_i\lambda_i$.

As we prove later, all the objects that we argue about are such that whenever an execution is not saturated and does not reach the object limit, any process p_l, meeting the requirements of Definition 2, would have an events sequence γ as required. For any such object implementation, our proofs construct set S_k only if the object limit is not reached by any S_{k-1}-execution. Thus, whenever we need to construct S_k, any non-saturated execution in S_{k-1} would have the required γ.

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.

Definition 2 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 $b \in \text{poly}(n)$, 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 1 below 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 the proof is in increasing L, which later translates to higher lower bounds. The specific bounds obtained in Lemma 1 are summarized in Table 1.

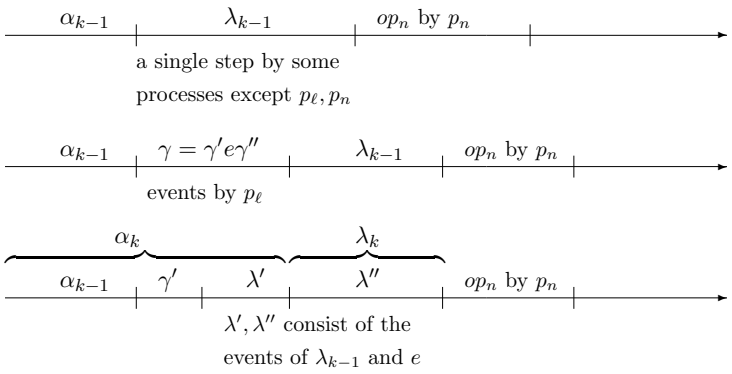

Figure 2: An L-perturbable execution: op_n performs different sequences of events after $\alpha_{k-1}\lambda_{k-1}$ and after $\alpha_k\lambda_k$. Event e is the first event of γ such that op_n's sequences of events after $\alpha_{k-1}\gamma' e\lambda_{k-1}$ and after $\alpha_{k-1}\lambda_{k-1}$ differ.

Lemma 1 *1. A b-bounded-value max register is $(b-1)$-perturbable.*

2. An m-limited-use max register is $(m-1)$-perturbable.

3. An m-limited-use counter is $(\sqrt{m}-1)$-perturbable.

4. A k-additive-accurate m-limited-use counter is $(\sqrt{\frac{m}{k}}-1)$-perturbable.

5. An m-limited-use b-valued compare-and-swap object is $(m/2-1)$-perturbable (if $b \geq n$).

6. 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 (2) 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, $r > 0$, 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 an r-perturbing execution may be constructed from $\alpha_{r-1}\lambda_{r-1}$ and γ as specified by the third case of 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} are going to be 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

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

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 $(m/2 - 1)$-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 $r < m/2 - 1$ 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 (2) of Definition 2, it is also an r-perturbing execution.

   Otherwise, our induction hypotheses are the following.

   (a) Execution $\alpha_{r-1}\lambda_{r-1}$ includes at most $2(r - 1)$ *CAS* operation instances, at most two instances performed by any single process, and all these instances are of the form $CAS(i,i + 1)$, for some $0 \leq i < r$.

   (b) Let k be the largest integer such that one or more $CAS(k,k + 1)$ instances are included in $\alpha_{r-1}\lambda_{r-1}$, then there is a successful $CAS(k - 1,k)$ instance in α_{r-1}.

   These two properties imply that a *read* after $\alpha_{r-1}\lambda_{r-1}$ returns either k or $k + 1$: $\alpha_{r-1}\lambda_{r-1}$ includes a successful $CAS(k-1,k)$, it includes instances of $CAS(k,k+1)$ (which may complete successfully or not) but of no higher value, and all *CAS* instances it includes increment the value.

   We assume the initial value of the CAS object is 0 and define the value k associated with $\alpha_{r-1}\lambda_{r-1}$ when $r = 0$ as 0. We note that induction hypotheses a) and b) above are vacuously satisfied for $r = 0$.

   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 performs a CAS$(k,k + 1)$ operation instance

after α_{r-1}. Since $r < m/2 - 1$, it follows from the fact that $\alpha_{r-1}\lambda_{r-1}$ is not saturated and from the induction hypotheses that $k + 1 < b$.

There are two possibilities:

- There is an event e such that $\gamma = \gamma' e \gamma''$ and e is the first event of γ such that the sequences of events taken by p_n as it performs a *read* operation by itself after $\alpha_{r-1}\lambda_{r-1}$ and after $\alpha_{r-1}\gamma' e \lambda_{r-1}$ differ. It follows that an r-perturbing execution may be constructed from $\alpha_{r-1}\lambda_{r-1}$ and γ as specified by the third case of Definition 2. Since p_ℓ performs a single $CAS(k,k+1)$ instance in γ, properties (a) and (b) hold also after the resulting r-perturbing execution.
- Otherwise, the value returned by op_n's *read* operation is the same when it executes after $\alpha_{r-1}\lambda_{r-1}$ and after $\alpha_{r-1}\gamma\lambda_{r-1}$. We claim that op_n's *read* operation after $\alpha_{r-1}\gamma\lambda_{r-1}$ returns $k+1$. This is clearly the case if the CAS$(k,k+1)$ instance by p_ℓ is successful. If the CAS$(k,k+1)$ instance fails, then it follows from the induction hypothesis b) that a successful CAS$(k,k+1)$ instance by another process must have been linearized in $\alpha_{r-1}\gamma$ and so op_n's *read* operation after $\alpha_{r-1}\gamma$ must return $k+1$, also in this case.

  We extend γ by an execution fragment γ', in which p_ℓ performs a second operation—a $CAS(k+1, k+2)$ instance. Since this is the first $CAS(k+1, k+2)$ instance in $\alpha_{r-1}\gamma\gamma'$, it follows that p_n's *read* operation returns $k+2$ after $\alpha_{r-1}\gamma\gamma'$ whereas it returns k or $k+1$ after α_{r-1}. We can therefore construct an r-perturbing execution from $\alpha_{r-1}\lambda_{r-1}$ and $\gamma\gamma'$, as specified by the third case of Definition 2. Since p_ℓ performs in $\gamma\gamma'$ one $CAS(k,k+1)$ instance and one $CAS(k+1,k+2)$ instance, properties (a) and (b) hold also for the resulting r-perturbing execution.

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 (2) 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 $update(v'_\ell)$ 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 an r-perturbing execution may be constructed from $\alpha_{r-1}\lambda_{r-1}$ and γ as specified by the third case of Definition 2.

■

4 Time Lower Bounds for Deterministic L-Pertubable Objects

In this section, we prove lower bounds for obstruction-free implementations of some well-known restricted-use objects.

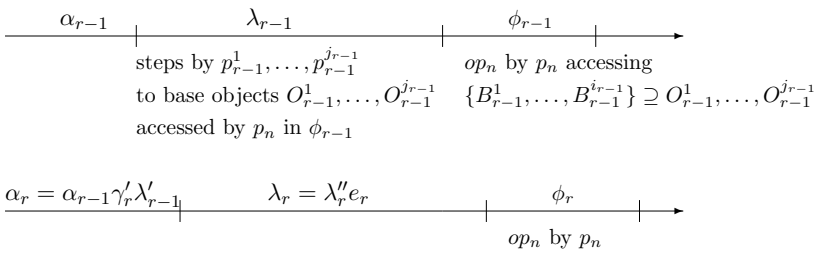

Figure 3: The structure of executions $\alpha_r\lambda_r\phi_r$, for $r \in \{0, \ldots, L\}$, constructed by the proof of Theorem 2. The notations op_n, α_r, λ_r, ϕ_r are as in Definition 2. The notation e_r refers to the event e defined in the third case of Definition 2, used for constructing execution $\alpha_r\lambda_r\phi_r$. The execution $\alpha_r\lambda_r$ is p_n-free for every r.

4.1 Lower bounds for implementations using historyless objects

We start by proving a step lower bound for L-perturbable implementations, in Theorem 2 below. The proof constructs a series of executions $\{\alpha_r\lambda_r\phi_r\}_{r=0}^{L}$, whose structure is depicted in Figure 3. (See Lemma 3 for a formal specification of these executions.) In the r'th execution of the sequence, process p_n accesses objects $B^1_r, \ldots, B^{i_r}_r$, while executing ϕ_r. Exactly j_r of these objects, denoted by $O^1_r, \ldots O^{j_r}_r$, are about to be written, right after α_r, by processes $p^1_r, \ldots, p^{j_r}_r$.

Our goal is to prove that p_n has to access a large number of base objects as it runs solo while performing an instance op_n of Op in one of the executions $\{\alpha_r\lambda_r\phi_r\}_{r=0}^{L}$. As we prove, if λ_r consists of steps by $n-1$ distinct processes, then the lower bound is linear in the number of processes, otherwise it is logarithmic in L.

To construct $\alpha_r\lambda_r\phi_r$, we deploy a free process, p_{ℓ_r} and let it run solo until it is about to perform a nontrivial event, e_r, to an uncovered object, along π_r. Let $\pi_r =< B^1_r, \ldots, B^{i_r}_r >$ 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 [7, 13]. 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 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. $\Psi(r)$ is a $(\log L)$-digit binary number defined as follows. Bit 0 (the most significant bit) of $\Psi(r)$ 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 $\Psi(r)$ 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.

As mentioned before, we construct the r'th sequence execution by deploying a free process, p_{ℓ_r} and letting it run solo until it is about to write to an uncovered object, B, along π_r. In terms of Ψ, this implies that the covering event e_r might flip some of the digits of $\Psi(r-1)$ from 1 to 0. But B 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 $\Psi(r)$ increases. It follows that $\Psi(r) = L-1$ must eventually hold, implying that π_r's length is $\Omega(\log L)$.

Theorem 2 *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: To prove the theorem, we construct a sequence of L executions and show that the lower bound is attained in one of them. The following lemma specifies the properties of executions in this sequence and proves that every implementation of an L-perturbable object has such a sequence.

Lemma 3 *The implementation has a sequence of executions $\{\alpha_r\lambda_r\phi_r\}_{r=0}^{L}$, defined as follows. The execution $\alpha_0\lambda_0$ 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 $j_r \geq 0$ events by j_r distinct processes, $p_r^1, \ldots, p_r^{j_r}$, applying nontrivial operations 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. *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 $\phi_r = \phi_{r-1}$. 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 after $\alpha_{r-1}\lambda_{r-1}$ 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 $\{B_{r-1}^1, \ldots, B_{r-1}^{i_{r-1}}\} \setminus \{O_{r-1}^1, \ldots, O_{r-1}^{j_{r-1}}\}$.*

Proof: 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 defined above, 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 (2) of Definition 2, $\alpha_r = \alpha_{r-1}$, $\lambda_r = \lambda_{r-1}$ and $\alpha_r\lambda_r$ is r-perturbing. Moreover, by the first case of Property 4 specified by the lemma, $\alpha_r\lambda_r\phi_r$ is the r'th sequence execution, where $\phi_r = \phi_{r-1}$.

Assume otherwise. Then, by case (3) 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_r such that the sequences of events taken by p_n as it performs op_n after $\alpha_{r-1}\lambda_{r-1}$ and after $\alpha_{r-1}\gamma_r'e_r\lambda_{r-1}$ differ. 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 performs different sequences of events 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 required by the lemma.

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 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.

By construction, $\alpha_r\lambda_r$ is r-perturbing (Definition 2).

By construction, $\alpha_r\lambda_r$ is p_n-free (Property 1), and ϕ_r is a solo execution fragment by p_n in which it performs op_n (Property 2). 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 now prove that the lower bound is attained in one of the executions $\alpha_r\lambda_r\phi_r$, $r \in \{0, \ldots, L\}$.

If $\alpha_r\lambda_r\phi_r$ is saturated, for some $r \in \{0, \ldots, L\}$, then Property 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 $\{\alpha_r\lambda_r\phi_r\}_{r=0}^{L}$.

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, and let S_{π_r} denote the set of these base objects. Let $S^C_r = \{O^1_r, \ldots, O^{j_r}_r\}$ be the set of base objects defined in Property 3. Observe that, by Property 3, $S^C_r \subseteq S_{\pi_r}$ holds. Without loss of generality, assume that $O^1_r, \ldots, O^{j_r}_r$ 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^j_r be the indicator variable whose value is 1 if $B^j_r \in S^C_r$ and 0 otherwise. We associate an integral progress parameter, $\Psi(r)$, with each $r \geq 0$, defined as follows:

$$\Psi(r) = \sum_{j=1}^{i_r} b^j_r \cdot \frac{L}{2^j} \ .$$

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 $\Psi(r)$ 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^C_r, or 0 otherwise. This implies that the number of 1-bits in $\Psi(r)$ equals $|S^C_r|$. Our execution is constructed so that $\Psi(r)$ 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 the second case of Property 4, 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 $\{B^1_{r-1}, \ldots, B^{i_{r-1}}_{r-1}\} \setminus \{O^1_{r-1}, \ldots, O^{j_{r-1}}_{r-1}\}$. By Property 2, 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^k_{r-1}. This implies that $B^k_{r-1} \in S_{\pi_{r-1}} \setminus S^C_{r-1}$.

As $B^k_{r-1} \notin S^C_{r-1}$, we have $b^k_{r-1} = 0$. Since e_r is the first nontrivial event of p_{ℓ_r} in $\gamma'_r e_r$ to a base object in $S_{\pi_{r-1}} \setminus S^C_{r-1}$, we have that the values of objects $B^1_{r-1} \cdots B^{k-1}_{r-1}$ are the same after $\alpha_{r-1}\lambda_{r-1}$ and $\alpha_r \lambda_r$. It follows that $b^j_{r-1} = b^j_r$ for $j \in \{1, \ldots, k-1\}$. This implies, in turn, that $B^k_{r-1} = B^k_r$. As $B^k_r \in S^C_r$, we have $b^k_r = 1$. We get:

$$\begin{aligned}
\Psi(r) &= \sum_{j=1}^{i_r} b^j_r \cdot \frac{L}{2^j} \\
&= \sum_{j=1}^{k-1} b^j_r \cdot \frac{L}{2^j} + b^k_r \cdot \frac{L}{2^k} + \sum_{j=k+1}^{i_r} b^j_r \cdot \frac{L}{2^j} \\
&= \sum_{j=1}^{k-1} b^j_{r-1} \cdot \frac{L}{2^j} + \frac{L}{2^k} + \sum_{j=k+1}^{i_r} b^j_r \cdot \frac{L}{2^j} \\
&\geq \sum_{j=1}^{k-1} b^j_{r-1} \cdot \frac{L}{2^j} + \frac{L}{2^k} \\
&> \sum_{j=1}^{k-1} b^j_{r-1} \cdot \frac{L}{2^j} + \sum_{j=k+1}^{i_{r-1}} b^j_{r-1} \frac{L}{2^j} \\
&= \Psi(r-1),
\end{aligned}$$

where the last equality is based on the observation that $b^k_{r-1} = 0$.

As $\Psi(0) = 0$ and since $\Psi(r)$ strictly grows with r and can never exceed $L-1$, it follows that $\Psi(L) = L - 1$, which concludes the proof. ■

Lemma 1 and Theorem 2 imply the following step lower bounds (the bounds are also summarized in Table 1).

Theorem 4 *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.

4.2 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 its time performance. 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.* [10] 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. In their model, an event e applied by a process p to object O in an execution α *incurs* k *memory stalls* if k events are applied to the object by distinct processes while e is pending. This definition depends on modeling concurrency explicitly, and is a little awkward to work with if we model concurrency by interleaving. However, we can treat such an event as incurring k memory stalls if it is immediately preceded by a sequence of events by distinct processes different from p that include k events that apply nontrivial primitives to O. The intuition for why this works is that any such sequential schedule can be reordered to produce a concurrent schedule with k memory stalls by the Dwork *et al.* definition.

Our next result shows a lower bound on implementations using *arbitrary* read-modify-write primitives. The proof of Theorem 2, presented earlier, uses a 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 following proof constructs a series of executions, in which each new execution deploys a process that covers *some* (not necessarily uncovered) object along p_n's path.

In the r'th execution of the sequence, process p_n accesses objects $B_r^1, \ldots, B_r^{i_r}$. On some of these objects, denoted by $O_r^1, \ldots O_r^{j_r}$, p_n incurs memory stalls. If the number of memory stalls incurred by p_n in the r'th execution equals $n-1$, then the execution is saturated; as in Definition 2, we assume in this case for presentation simplicity that the rest of the sequence's executions are identical to the r'th execution (see Property 4 and the first sentence of Property 5 in the statement of Lemma 6 below).

Otherwise, we may deploy another process and let it run until about to write to some object along p_n's path (see the second sentence of Property 5 in Lemma 6). Let B_{r-1}^k denote this object. Then the prefix of p_n's path up to (and including) B_{r-1}^k does not change, but the rest of the path may change since p_n may read a different value when it accesses B_{r-1}^k.

We prove a logarithmic lower bound on the time complexity of obstruction-free implementations of L-perturbable objects from arbitrary primitives. Specifically, we prove that for such implementations either the step-complexity or the memory-stalls complexity is $\Omega(\min(\log L, n))$.

The proof employs a variation of the backtracking covering technique. We remind the reader that the proof of the step lower bound on implementations from historyless primitives (Theorem 2) constructed a sequence of executions and used a progress function Ψ, assigning integral values to these executions. The function Ψ was a monotonically-increasing progress function of r and $\Psi(r)$ could be viewed as a $(\log L)$-digit binary number whose bit i equals 1 if and only if the i'th object in p_n's path was covered after the r'th execution.

The function Ψ cannot be used for proving a time lower bound on implementations using arbitrary primitives, since up to $n-1$ memory stalls may be incurred by p_n as it accesses a single object. Consequently, the proof that follows employs a different progress function Φ, where $\Phi(r)$ can be viewed as an s-digit number in base n, where $L = 2^{2s}$. (Note that we may assume that the path taken by p_n always consists of at most s distinct objects, since otherwise the proof easily follows.) Here, the i'th digit of $\Phi(r)$ represents the number of memory stalls that will be incurred by p_n as it accesses the i'th object in its path in the course of the r'th sequence execution.

Similarly to the proof of Theorem 2, we show that Φ is a monotonically-increasing function of r, implying that all the values $\Phi(1), \ldots, \Phi(L)$ are distinct. We then use a "bins-and-balls" argument establishing that the value of at least one of the digits of $\Phi(1), \ldots, \Phi(L)$ must be at least s, implying

that p_n incurs at least s memory stalls when it accesses the corresponding objects.

Theorem 5 *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. Assume otherwise, then our proof constructs a sequence of L executions and shows that the lower bound is attained in one of them. The next lemma specifies the properties of executions in this sequence and proves that the implementation has such a sequence.

Lemma 6 *The implementation has a sequence of executions $\{\alpha_r\sigma_{r,1}\cdots\sigma_{r,j_r}\rho_r\}_{r=0}^{L}$, defined as follows. The execution 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 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 $|S_r^{j_r}| = |S_{r-1}^{j_r}| + 1$.*

Proof: 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, $0 \leq r \leq L$.

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 sequence execution, $\alpha_{r+1}\sigma_{r+1,1}\cdots\sigma_{r+1,j_{r+1}}\rho_{r+1}$.

By the 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 the induction

hypothesis, Definition 2, and the third property specified by the lemma, $\alpha_{r+1}\sigma_{r+1,1}\cdots\sigma_{r+1,j_{r+1}}\rho_{r+1}$ is the $(r+1)$'th sequence 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 the sequences of events taken by p_n as it performs op_n by itself after $\alpha_r\lambda_r$ and after $\alpha_r\gamma'_{r+1}e_{r+1}\lambda_r$ differ. 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 fragment in which process p_n runs solo after $\alpha_{r+1}\sigma_{r+1,1}\cdots\sigma_{r+1,j_{r+1}}$ until it completes the operation instance op_n, in the course of which it accesses the base objects $B_{r+1}^1,\ldots,B_r^{i_{r+1}}$. It follows from our construction that $\alpha_{r+1}\sigma_{r+1,1}\cdots\sigma_{r+1,j_{r+1}}\rho_{r+1}$ is the $(r+1)$'th execution in sequence. ■

If one of these executions, $\alpha_r\sigma_{r,1}\cdots\sigma_{r,j_r}\rho_r$, for some $r \leq L$, is saturated, then it follows from from the third property specified by the lemma 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 constructed by the lemma is saturated. We will prove that p_n incurs $\Omega(s)$ memory stalls in one of these executions.

We remind the reader that we let B_r^i denote that i'th object accessed by p_n in $\alpha_r\sigma_{r,1}\cdots\sigma_{r,j_r}\rho_r$, and we let O_r^m and S_r^m respectively denote the m'th covered object along p_n's path and the set of processes that cover it (see the statement of Lemma 6).

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

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

Let $N_r = \sum_{j=1}^{i_r} n_r^j$. Thus, it suffices to show that one of these executions has $N_r = \Omega(s)$. We

associate the following integral progress parameter, $\Phi(r)$, with each execution $r \geq 0$:

$$\Phi(r) = \sum_{i=1}^{i_r} n_r^i \cdot n^{s-i}. \tag{2}$$

$\Phi(r)$ can be viewed as an s-digit number in base n 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 required by Lemma 6, $O_{r+1}^{j_{r+1}} = B_r^k$, for some $1 \leq k \leq i_r$ 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 $i \in \{k+1, \ldots, i_r\}$. We get:

$$\begin{aligned}
\Phi(r+1) &= \sum_{i=1}^{i_{r+1}} n_{r+1}^i \cdot n^{s-i} \\
&= \sum_{i=1}^{k} n_{r+1}^i \cdot n^{s-i} \\
&= \sum_{i=1}^{k-1} n_r^i \cdot n^{s-i} + (n_r^k + 1) \cdot n^{s-k} \\
&> \sum_{i=1}^{k} n_r^i \cdot n^{s-i} + \sum_{i=k+1}^{s} (s-1) \cdot n^{s-i} \\
&\geq \sum_{i=1}^{i_r} n_r^i \cdot n^{s-i} \\
&= \Phi(r)
\end{aligned}$$

Since the sequence of values $\Phi(1), \ldots, \Phi(L)$ is strictly increasing, each $\Phi(r)$ is unique. By the definition of Φ, each value $\Phi(r)$ corresponds to a different partitioning of integer N_r to the values of the s digits of $\Phi(r)$. What is the maximum number $\mathcal{N}$ of different executions r for which $N_r \leq s$ 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:

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

Where the penultimate 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 [11, page 75]. Thus, for all $L \geq 4$, there is an execution $\alpha_{r'}\sigma_{r',1} \cdots \sigma_{r',j_{r'}}\rho_{r'}$ such that $N_{r'} > s$ holds. ∎

From Theorem 5 and Lemma 1, we obtain the following specific bounds (see also in Table 1).

Theorem 7 *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.*

5 Space lower bounds for implementations using historyless objects

In this section we prove space lower bounds on L-perturbable objects. Although our proofs assume deterministic algorithms, they also apply for randomized algorithms. A randomized algorithm may make a random selection (often called a *coin-flip*) for determining what next step to take. Since a randomized algorithm can be seen as a weighted average of deterministic ones, space lower-bounds proven for deterministic algorithms apply also for randomized algorithms.

Theorem 8 *Let A be an n-process obstruction-free implementation of an object $\mathcal{O}$ from historyless primitives. Then A has an execution in which L distinct base objects are accessed.*

Proof: To prove the lower bound, we construct a sequence of L executions and show that many distinct objects are being written in them.

Lemma 9 *The implementation has a sequence of executions $\{\alpha_r\lambda_r\phi_r\}_{r=0}^{L}$ defined as follows. Execution $\alpha_0\lambda_0$ is empty, ϕ_0 is an execution of op_n by p_n, and for every r, $1 \leq r \leq L$, the following 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 op_n.*
3. *In λ_r, distinct processes $q_1, \ldots, q_r$ each apply a nontrivial operation to distinct base objects $O_1, \ldots, O_r$, respectively.*
4. *$|active(\alpha_r\lambda_r)| \leq r$.*

Proof: 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 properties specified by the lemma, 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 (3) 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_r such that the sequences of events taken by p_n as it performs op_n by itself after $\alpha_{r-1}\lambda_{k-1}$ and after $\alpha_{r-1}\gamma'_r e_r \lambda_{r-1}$ differ. 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 required by the lemma. 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 2 and 3 are immediate from our construction.

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

The space lower bound follows immediately from Property 3 proved in the lemma. ■

The following space lower bounds on specific restricted-use objects follow immediately from Lemma 1 and Theorem 8 (see also in Table 1). As argued above, these lower bounds apply for both deterministic and randomized algorithms.

Theorem 10 *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, for $b \geq n$, from historyless primitives is $\Omega(\min(m, n))$.
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 $\Omega(\min(\sqrt{\frac{m}{k}}, n))$.

6 Lower Bounds for Randomized Implementations

Proving step lower bounds for *randomized* implementations of concurrent objects is more difficult, due to the extra flexibility these implementations have. However, lower bounds on many objects can be obtained using a variation of the L-perturbability argument that generalizes the lower bounds for max registers from [3] and for approximate counters from the conference version of the present work [5].

The basic idea is to construct a family of executions where instead of delaying operations as soon as they cover a register observed by the reader, we delay each step with some fixed probability q. Then on average each register contains only $1 + 1/q$ distinct values across the family of executions, as compared to at most 2 in the deterministic case. This gives a bound of $(1 + 1/q)^r$ on the expected number of distinct values that can be returned by a deterministic reader that runs for r steps. Constructing a family of executions for a specific object that forces the reader to return many values on average despite delayed operations gives the lower bound.

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 [19]. For simplicity, we provide a fixed operation to each process and use the oblivious adversary only to specify the timing of process steps. 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.

Our main tool is a variant of L-perturbability that we call *uniform* (p, L)*-perturbability*. As with L-perturbability, our goal is to generate a family of executions in which op_n returns $L + 1$ distinct values. But instead of generating these executions by repeated perturbations of previous executions, we consider a *single* sequence of update operations $\gamma_1 \dots \gamma_{n-1}$, choose each update to include a delayed step with independent probability at most p, and then consider all n executions obtained by taking the first j updates followed by any delayed operations and op_n. The idea is that an implementation is uniformly (p, L)-perturbable if the resulting family of executions always yields at least $L+1$ distinct values on average, for any choice of which specific step of each delayed operation to delay (a formal definition appears in Definition 3.)

The intuition is that for many common objects, if p is small enough, most of the update operations are completed, so as more and more of these are included in the execution, any implementation of the object is forced to return new values from op_n no matter how the smaller number of delayed operations are linearized. This intuition is justified for many common objects in Lemma 11.

If each update γ_i has expected step complexity w, then delaying each operation with probability p/w satisfies the requirements of uniform (p, L)-perturbability while preventing any historyless base object from taking on more than $O(w/p)$ distinct values on average. This means that an implementation of op_n that accesses at most r base objects will see at most $O((w/p)^r)$ distinct sequences of values, which gives (Lemma 12) an upper bound on the expected value of L. The same lemma includes similar bounds for general objects with bounded contention. The collision between these upper bounds and the lower bound in the definition of uniform (p, L)-perturbability gives the full lower bounds, stated in Theorem 15.

6.1 Uniform perturbability

Definition 3 *An implementation $\mathcal{I}$ is* uniformly (p, L)-perturbable *if there is an operation instance op_n by process p_n and operation instances $\gamma_1 \dots \gamma_k$ by distinct processes $p_1 \dots p_k$, with $k < n$, such that, if for each i, with independent probability at most p, $\gamma'_i\delta_i$ is a prefix of γ_i where δ_i is a single step, and $\gamma'_i = \gamma_i$ and δ_i is the empty sequence otherwise, then in the family of executions $\Xi_i = \gamma'_1 \dots \gamma'_i\delta_i \dots \delta_1 op_n$, where $0 \le i \le k$, the set of values returned by op_n contains at least $L + 1$ distinct elements on average.*

An object is *uniformly* (p, L)*-perturbable* if all its obstruction-free implementations are uniformly (p, L)-perturbable.

The $+1$ in the definition is for consistency with L-perturbability; a family of executions yields $L + 1$ distinct values if the return value changes L times.

6.2 Uniform perturbability of particular objects

For typical restricted-use objects, our goal will be to show uniform (p, L)-perturbability where p is a constant and L is polynomial in n. For asymptotic bounds on step complexity, the exact value

of p and the exponent on L only affect the constants, so in the lemma below these are chosen for simplicity rather than optimality.

Lemma 11 *1. A b-bounded-value max register is uniformly $(1/4, \Omega(\min(b,n)))$-perturbable.*

2. An m-limited-use max register is uniformly $(1/4, \Omega(\min(m,n)))$-perturbable.

3. A c-multiplicative-accurate m-limited-use counter is uniformly $(1/4, \Omega(\log\min(m,n)/\log c))$-perturbable.

4. The activity counter of Bender and Gilbert [9] is $(1/4, \Omega(\log n))$-perturbable.

5. An m-limited-use counter is uniformly $(1/4, \Omega(\log\min(m,n)))$-perturbable.

6. A k-additive-accurate m-limited-use counter is uniformly $(1/4, \Omega(\log\min(m/k,n)))$-perturbable.

7. An m-limited-use b-valued compare-and-swap object is uniformly $(1/4, \Omega(\min(\sqrt{m}, b, \sqrt{n})))$-perturbable.

8. An m-limited-use collect object is uniformly $(1/4, \Omega(\min(m,n)))$-perturbable.

Proof: 1. Let γ_i = `Write`(i) for $1 \le i \le \min(b-1, n-1)$ and let op_n = `ReadMax`(). For each γ_i that is not truncated, the corresponding execution $\Xi_i = \gamma_1 \ldots \gamma_i \delta_i \ldots \delta_1 op_n$ contains a complete `Write`(i) operation that must be linearized before op_n and no `Write`(i') operation for any $i' > i$. It follows that op_n returns i in this execution. Since on average $(1-p)\min(b-1, n-1)$ operations are not truncated, op_n returns $\Omega(\min(b,n))$ distinct values on average.

2. Follows from the same construction as for a b-bounded counter.

3. Let each γ_i, for $1 \le i \le \min(m, n-1)$, be a `CounterIncrement` and let op_n 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 $m_i = \left\lfloor (2c)^{2i} \sqrt{\min(m, n-1)} \right\rfloor$ calls to `CounterIncrement`, where i ranges from 0 to $\ell - 1 = \left\lfloor \frac{1}{2} \log_{2c} \sqrt{\min(m, n-1)} \right\rfloor - 1 = \Theta(\log\min(m,n)/\log c)$.

   Let T_i be the number of truncated increments among the first m_i increments. Since each increment is truncated with probability at most $1/4$, we have $\mathrm{E}[T_i] \le m_i/4$. Standard Chernoff bounds (for example, [18, (4.2)]) give, for $0 \le \delta \le 1$,

$$\Pr[T_i \ge (1+\delta) m_i/4] \le e^{-(m_i/4)\delta^2/3} \tag{3}$$

   Let a be a constant, and let $\delta = \sqrt{\frac{24a \ln m_i}{m_i}} = o(1)$. Then (3) becomes

$$\begin{aligned} \Pr[T_i \ge m_i (1/4 + o(1))] &\le e^{-a \ln m_i} \\ &= m_i^{-2a}. \end{aligned}$$

   Because $m_i = \Omega\left(\sqrt{\min(m, n-1)}\right)$, the error probability is $\Omega(\min(m, n-1)^{-a})$, and applying the union bound gives that $T_i < (m_i)(1/4 + o(1))$

for all $\ell \ll \min(m, n-1)$ executions with probability at least $1 - \Omega\left(\min(m, n-1)^{-a+1}\right)$.

Suppose that this event holds. Let v_i be the number of increments that can be linearized before the `CounterRead` in the execution Ξ_{k_i}. Then we have $m_i/2 < m_i(3/4 - o(1)) < v_i \leq m_i$, and for the return value x_i of a correct c-multiplicative-accurate `CounterRead` we have $m_i/2c < x_i \leq cm_i$. From our definition of m_i, we have $m_i = m_{i+1}/4c^2$, so $x_i \leq cm_i = cm_{i+1}/4c^2 = m_i/4c^2 < x_{i+1}$. It follows that the return value of the `CounterRead` in any two distinct executions Ξ_{k_i}, Ξ_{k_j} is distinct. We thus get ℓ distinct values with high probability, giving at least $\ell/2$ distinct values on average for sufficiently large m and n, which is $\Omega(\log \min(m, n)/\log c)$.

4. Recall that the activity counter of [9] differs from an idealized c-multiplicative-accurate counter in that `CounterRead` may return inaccurate values with probability n^{-a} for any fixed a and sufficiently large n after any number of increments, and with larger probability after fewer than $O(\log^4 n)$ increments. So with high probability, of the executions Ξ_{k_i} constructed above, only those with $i = O(\log \log n)$ will return different values from an activity counter as from a c-multiplicative-accurate counter. This leaves $\ell/4 - O(\log \log n) = \Omega(\log n)$ distinct values, giving the claimed bound.

5. Immediate from the c-multiplicative-accurate bound.

6. Immediate from the c-multiplicative-accurate bound.

7. Let $\ell = \left\lfloor \sqrt{\min(m, (b-1)^2, n-1)} \right\rfloor$ and let $k = \ell^2$. Construct the sequence $\gamma_1 \dots \gamma_k$ by concatenating ℓ blocks of ℓ CAS operations each, where the i-th block for $i = 1 \dots \ell$ consists entirely of $CAS(i-1, i)$ operations.

   Observe that the probability that a single block contains no undelayed operation is at most $(1/4)^\ell$. By the union bound, the probability that every block contains at least one complete CAS operation is at least $1 - \ell(1/4)^\ell$. As this goes to 1 in the limit, for sufficiently large ℓ we have that it is at least $1/2$. If this event occurs, we get a sequence of truncated operations $\gamma'_1 \dots \gamma'_k$ that includes complete operations $CAS(0,1)CAS(1,2)\dots CAS(\ell-1, \ell)$ as a subsequence. It follows that between them the executions $\Xi_1 \dots \Xi_k$ cause the reader to return $\ell + 1$ distinct values. Taking the expectation gives $L \geq (1/2)(\ell+1) = \Omega\left(\min\left(\sqrt{m}, b, \sqrt{n}\right)\right)$.

8. By reduction from an m-limited-use max register: implement each `Write`(i) operation by p_i by a store operation, and implement `ReadMax` by performing a single collect from which we compute the maximum value. Then apply the same construction as for a max register.

■

6.3 Lower bound for uniformly perturbable objects

We now state the core lemma for our lower bound for uniformly (p, L)-perturbable objects.

Lemma 12 *Let $\mathcal{O}$ be uniformly (p, L)-perturbable. For any obstruction-free implementation of $\mathcal{O}$ in which each update operation takes at most w steps on average and op_n takes at most r steps always, we have*

$$(w/p)(1 + (C+1)(w/p))^r - 1 \geq L \tag{4}$$

if the implementation uses base objects with maximum contention C, and

$$(w/p)(1+(w/p))^r - 1 \geq L \tag{5}$$

if the implementation uses historyless base objects.

6.3.1 Proof of Lemma 12

Given a uniformly (p, L)-perturbable object where each update operation runs in expected w steps, let $\gamma_1 \dots \gamma_k$ be a sequence of perturbing operations and op_n an operation that together demonstrate (p, L)-perturbability. We will construct a family of executions Ξ_i in which each γ_i operation is delayed with probability at most p, by delaying each step of γ_i with independent probability $q = p/w$. We will then show that unless the bounds in the lemma hold, these executions do not allow op_n to return enough distinct values on average to satisfy the requirement of (p, L)-perturbability.

Let j_i be the number of steps of γ_i that are not delayed; observe that j_i is a random variable with geometric distribution. Construct a family of schedules σ_{ij} of the form

$$\sigma_{ij} = \underbrace{p_1^{j_1} p_2^{j_2} \dots p_i^{j}}_{\gamma'_1 \dots \gamma'_i} \underbrace{p_i p_{i-1} \dots p_1}_{\delta_i \dots \delta_1} \underbrace{p_n^r}_{op_n},$$

where $1 \leq j \leq j_i$ and p_i^x represents x many steps by p_i. Note that this schedule may assign steps to processes that have already completed their operations; we assume that any such steps become no-ops. For the special case $i = 0$, let σ_{00} be the schedule p_n^r. Let Ξ_i, for each i in $0 \dots k$, be the execution corresponding to σ_{ij_i}.

Each operation γ_i includes at most w steps on average; each step is delayed with probability p/w. So by Wald's lemma, the probability that γ_i includes a delayed step is at most $w(p/w) = p$. The construction thus satisfies the requirements of Definition 3, so the executions Ξ_i cause op_n to return $L + 1$ values on average.

Given a schedule σ_{ij}, let $B_1, \dots B_r$ be the base objects accessed by the first, second, etc., steps of p_n. Define $c_{ij}(k)$ to be the number of delayed operations δ in σ_{ij} that access B_k. Define $b_{ij}(k)$ to be 1 if some delayed operation δ applies a non-trivial primitive to B_k and 0 otherwise.

The significance of these quantities is that $c_{ij}(k)$ measures the number of simultaneous primitives on B_k if all operations $\delta_i \dots \delta_1$ are scheduled concurrently, while $b_{ij}(k)$ simply records whether B_k is covered by a primitive in $\delta_i \dots \delta_1$ or not. Both values are bounded: $c_{ij}(k)$ is at most $C + 1$ if we have a contention bound C, while $b_{ij}(k)$ is at most 1 by definition.

The following randomized progress measure is an adaptation of Ψ from Section 4.1. It is defined for any vector $c = c(1)c(2) \dots c(r)$ with $0 \leq c(i) \leq m$,

$$\Psi_{p,m}(c) = (1/p) \sum_{i=1}^{r} (1 + m/p)^{r-i} c(i). \tag{6}$$

Unlike Ψ, which increases by 1 whenever a new base object is covered, $\Psi_{p,m}$ only increases by 1 *in expectation* when some operation γ_i attempts to apply an operation to a base object observed by op_n. As these are the only events that can change the return value of op_n, the expected value of $\Psi_{p,m}$ is an upper bound on the maximum number of times op_n's view has changed. It follows that the maximum possible value of $\Psi_{p,m}$ bounds L. The intuition is that a step applied to the i-th register observed by op_n will not change $c(j)$ for $j < i$, will increase $c(i)$ by 1 with some probability,

and might change $c(j)$ arbitrarily for $j > i$, as observing the change in B_i may divert subsequent reads by op_n to different base objects that may or may not already be covered.

We begin by showing that changes to c_i following these rules do indeed have the claimed effect on $\mathrm{E}[\Psi_{p,m}]$.

Lemma 13 *Given a vector $c = c(1)c(2)\ldots c(r)$ with $0 \leq c(i) \leq m$, let c' be obtained from c by choosing some position i and letting*

1. *$c'(j) = c(j)$ for $j < i$;*
2. *$c'(i) = c(i) + X$, where X is a $0-1$ random variable that is 1 with probability p; and*
3. *$c'(j) \geq 0$ for all $j > i$.*

Then

$$\mathrm{E}[\Psi_{p,m}(c')] \geq \Psi_{p,m}(c) + 1. \tag{7}$$

Proof: First, we remove some of the dependence on r by a change of variables. Write $\Psi_{p,m}(c)$ as $\sum_{i=0}^{r-1} w_i c(r-i)$. We will show that $w_i = (1/p)(1+m/p)^i$ is the solution to a recurrence that arises from (7), which after undoing the change of variables yields the coefficients $(1/p)(1+m/p)^{r-i}$ in (6).

First, compute

$$\begin{aligned}\mathrm{E}\left[\Psi_{p,m}(c') - \Psi_{p,m}(c)\right] &\geq \mathrm{E}[w_i X] - \sum_{j=0}^{i-1} w_j c(r-j) \\ &\geq w_i p - \sum_{j=0}^{i-1} w_j m. \end{aligned} \tag{8}$$

We want this final quantity to be at least 1 for all i. Suppose that it is equal to 1, i.e.,

$$p w_i - m \sum_{j=0}^{i-1} w_j = 1. \tag{9}$$

Let $F(z) = \sum_{i=0}^{\infty} w_i z^i$ be the ordinary generating function for w_i. Then (9) can be rewritten as

$$pF(z) - m\frac{z}{1-z}F(z) = \frac{1}{1-z}.$$

Solving for $F(z)$ gives

$$\begin{aligned} F(z) &= \frac{1}{(1-z)\left(p - m\frac{z}{1-z}\right)} \\ &= \frac{1}{p\left(1 - z - (m/p)z\right)} \\ &= (1/p)\frac{1}{1-(1+m/p)z}. \end{aligned}$$

This gives $w_i = (1/p)(1+m/p)^i$ as claimed. ∎

From the recurrence (9) we can also calculate that

$$\sum_{j=0}^{i-1} w_j m = pw_i - 1$$
$$= (1 + m/p)^i - 1.$$

This gives an upper bound of $(1+m/p)^r - 1$ on $\Psi_{p,m}(c)$ for any vector c with r components $c(i) \leq m$.

Let V_{ij} be the number of distinct values returned by op_n in the executions corresponding to schedules σ_{11} through σ_{ij}. Observe that V_{k,j_k} gives an upper bound on the number of distinct values returned by op_n in the subset $\Xi_1, \Xi_2, \ldots \Xi_k$ of these executions. We will show that V_{ij} is bounded in expectation by $\Psi_{p/w,C+1}(c_{ij})$ for implementations with contention bound C and by $\Psi_{p/w,1}(b_{ij})$ for implementations from historyless objects, by showing that $\Psi_{p/w,C+1}(c_{ij}) - V_{ij}$ and $\Psi_{p/w,1}(b_{ij}) - V_{ij}$ are submartingales.[3]

Lemma 14 *Fix p, and fix an obstruction-free implementation of $\mathcal{O}$ from either bounded-contention or historyless objects, in which every execution of γ_i takes at most w steps on average and every executions of op_n takes at most r steps. Let $\Xi_1 \ldots \Xi_k$ be as defined above. Then the expected number of distinct values returned by op_n in these executions is at most*

$$(w/p)(1 + (C+1)w/p)^r - 1 \tag{10}$$

for base objects with contention at most C and

$$(w/p)(1 + w/p)^r - 1 \tag{11}$$

for historyless base objects.

Proof: Observe that in the initial schedule σ_{00}, $V_{00} = \Psi_{p/w,C+1}(c_{00}) = \Psi_{p/w,1}(b_{00}) = 0$. So the initial values $\Psi_{p/w,C+1}(c_{00}) - V_{00}$ and $\Psi_{p/w,1}(b_{00}) - V_{00}$ are both 0.

First let us look at the case of arbitrary base objects, and consider what happens as we extend a schedule σ_{ij} by one step. Let $c = c_{ij}$ be the vector of delayed operation counts preceding this step and let $b = b_{ij}$ be the vector of covering indicators. Let c' and b' be the corresponding vectors following the step; that is, $c' = c_{i'j'}$ and $b' = b_{i'j'}$ where $i' = i$ and $j' = j+1$ or $i' = i+1$ and $j' = 1$ in the case $j = j_i$. Let $V = V_{ij}$ and $V' = V_{i'j'}$ be the number of distinct values returned by op_n in the executions corresponding to schedules σ_{11} through σ_{ij} or $\sigma_{i'j'}$, respectively.

If the step does not access any base object accessed by op_n, then there is no change to c, b, or V. Conditioning on this even occurring, $\Psi_{p/w,C+1}(c') - V' = \Psi_{p/w,C+1}(c) - V$ and similarly $\Psi_{p/w,1}(b') - V' = \Psi_{p/w,1}(b) - V$.

Alternatively, if the step does access a base object accessed by op_n, it may be the case that op_n returns a new value, giving $V' = V + 1$. To compensate for this, we need the progress measure to rise as well. Let k be the index of the first step in op_n that observes this base object. Then for any $k' < k$, $B_{k'}$ is a distinct base object from B_k, so $c'(k') = c(k')$ and $b'(k') = b(k')$ for these positions.

For k itself, with probability p/w the new operation is delayed to become δ_i. In the case of an arbitrary base object, this gives a p/w chance that $c'(k)$ increases by one from $c(k)$. Furthermore,

[3] A submartingale is a process $X_0, X_1, X_2, \ldots$ where $\mathrm{E}[X_{i+1}|X_1 \ldots X_i] \geq X_i$. A simple induction shows that, in any submartingale, $\mathrm{E}[X_i] \geq \mathrm{E}[X_0]$ for all i.

the reader may react to the new operation—whether it is delayed or not—by accessing different base objects for operations $k' > k$. This may cause $c'(k')$ to be less than $c(k')$ for these values of k', but none of them can drop below 0.

In the case of a historyless base object, we need only consider the case where $b(k) = 0$, as otherwise the new step is covered by some previous delayed operation δ, and V and b do not change. In this case, there is a similar p/w chance that $b'(k)$ increases to 1, while $b'(k)$ may drop to 0 for $k' > k$.

For both classes of objects, if we condition on the operation being visible to op_n, the conditions of Lemma 13 apply to the appropriate progress function, so we have

$$\mathrm{E}[\Psi_{p/w,C+1}(c') - V'] \geq \mathrm{E}[\Psi_{p/w,C+1}(c) - V]$$

and

$$\mathrm{E}[\Psi_{p/w,1}(b') - V'] \geq \mathrm{E}[\Psi_{p/w,1}(b) - V],$$

which establishes the submartingale property. From this it follows that

$$\mathrm{E}[\Psi_{p/w,C+1}(c_{k,j_k}) - V_{k,j_k}] \geq 0$$

and

$$\mathrm{E}[\Psi_{p/w,1}(b_{k,j_k}) - V_{k,j_k}] \geq 0$$

It follows that in either case, $\mathrm{E}[V_{k,j_k}]$ is bounded by the maximum possible value of Ψ, giving the claimed result. ■

To complete the proof of Lemma 12, observe that Lemma 13 provides upper bounds on precisely the same quantity $\mathrm{E}[V_{j,j_k}]$ for which L is a lower bound.

6.4 Lower bound on worst-case expected step complexity

Lemma 12 gives a rather technical result, which is also constrained by the assumption that op_n always runs in a fixed number of steps. The following theorem removes this restriction and restates the lower bound in terms of worst-case expected step complexity for any operation on a (p, L)-perturbable object.

Theorem 15 *Let $\mathcal{O}$ be uniformly (p, L)-perturbable. Then:*

1. *For any implementation of $\mathcal{O}$ from base objects with maximum contention C, there is some operation with worst-case expected step complexity*

$$\Omega\left(\frac{\log pL}{\log\left((C+1)/p\right) + \log\log pL}\right).$$

2. *For any implementation of $\mathcal{O}$ from historyless base objects, there is some operation with worst-case expected step complexity*

$$\Omega\left(\frac{\log pL}{\log\left(1/p\right) + \log\log pL}\right).$$

Proof: Consider a family of executions as constructed for the proof of Lemma 12. Let w be the maximum expected step complexity of each operation γ, and let r be the worst-case expected step complexity of op_n. Construct an object $\mathcal{O}'$ from $\mathcal{O}$ by truncating any instance of op_n that takes more than $2r$ steps and having it return a default value. By Markov's inequality, op_n will be truncated at most half the time on average, so if $\mathcal{O}$ is (p, L)-perturbable, $\mathcal{O}'$ is $(p, L/2)$-perturbable, and our derived implementation has the property that op_n always finishes in $2r$ steps.

From Lemma 12, we have that

$$(w/p)(1 + (C + 1)(w/p))^{2r} - 1 \geq L/2 \tag{12}$$

or

$$(w/p)(1 + (w/p))^{2r} - 1 \geq L/2 \tag{13}$$

for implementations of $\mathcal{O}$ from C-bounded-contention or historyless objects, respectively.

Observe that the left-hand sides of (12) and (13) are increasing functions in both w and r. So if there is some value s such that setting both w and r to s makes either inequality false, we must have at least one of w and r greater than s to make it true.

Taking logs and performing some tedious calculations shows that (12) fails for

$$s = \frac{1}{5} \cdot \frac{\log(pL/2)}{\log((C+1)/p) + \log\log pL},$$

giving

$$\max(w, r) = \Omega\left(\frac{\log pL}{\log((C+1)/p) + \log\log pL}\right).$$

The bound for historyless base objects is obtained by setting $C = 0$. ■

For fixed p and $C > 0$, the bounds simplify to $\Omega(\log L/(\log\log L + \log C))$ and $\Omega(\log L/\log\log L)$, respectively.

Because L only appears within a logarithm, polynomial changes in L yield only constant-factor changes in the bounds. Using the bounds on L computed in Lemma 11, this gives a lower bound on worst-case expected step complexity of $\Omega\left(\frac{\log n}{\log\log n + \log C}\right)$ using C-bounded-contention base objects and $\Omega\left(\frac{\log n}{\log\log n}\right)$ for poly(n)-bounded-value max registers, poly(n)-limited-use max registers, poly(n)-limited-use poly(n)-valued compare-and-swap objects, and poly(n)-limited-use collect objects; and corresponding lower bounds of $\Omega\left(\frac{\log\log n}{\log\log\log n + \log C}\right)$ and $\Omega\left(\frac{\log\log n}{\log\log\log n}\right)$ on $\Theta(1)$-multiplicative-accurate poly(n)-limited-use counters, activity counters, poly(n)-limited-use counters, and k-additive-accurate m-limited-use counters with $m/k = \text{poly}(n)$.

7 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 and matches the previously-known lower bound proved using a specialized

argument [3]. While we are not aware of a sublogarithmic upper bound for approximate randomized counters, the weaker activity counters of [9] have step complexity $O(\log \log n)$, close to the lower bound of $\Omega\left(\frac{\log \log n}{\log \log \log n}\right)$. It is unclear whether the other lower bounds are tight and it would be interesting to investigate that. Another interesting research direction is to devise generic implementations for L-perturbable objects. This is of particular interest in the case of randomized implementations, in which case, it is important to study 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*, 59(1), Feb. 2012. Previous version in *PODC*, pages 36–45, 2009.

[4] 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.

[5] J. Aspnes, H. Attiya, K. Censor-Hillel, and D. Hendler. Lower bounds for restricted-use objects. In *Twenty-Fourth ACM Symposium on Parallel Algorithms and Architectures*, pages 172–181, June 2012.

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

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

[8] 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.

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

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

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

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

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

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

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

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

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

[18] M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005.

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