Tight Space Bounds for Memoryless Anonymous Consensus

Leqi Zhu*

Abstract

Numerous randomized n-process binary consensus algorithms use $\Omega(n)$ registers [AH90, AW92, CIL94, AC08]. The best existing space lower bound for randomized (and obstruction-free) algorithms is $\Omega(\sqrt{n})$ [FHS98] registers of unbounded size. Closing this gap is a longstanding open problem.

We take a step towards solving this problem by proving a lower bound of n for the space complexity of randomized (and obstruction-free) binary consensus in a restricted computational model. Specifically, we consider an asynchronous system with n anonymous processes, which communicate through an m-component multi-writer snapshot. Each process alternately performs SCAN and UPDATE, but does not remember information from one UPDATE to the next. Our lower bound uses a covering argument combined with an innovative valency argument.

We also provide a matching upper bound of n components in this model, which improves the space upper bounds given by the existing bounded space anonymous obstruction-free algorithms [GR05, Bow11], which use $8n + 2$ components and $2n$ components, respectively.

*Department of Computer Science, University of Toronto, Toronto, ON M5S 3G4, Canada (lezhu@cs.toronto.edu)

1 Introduction

Consensus is a fundamental problem in distributed computing. While it is impossible to solve consensus in a wait-free manner using only registers in a deterministic shared memory setting [FLP85], it is possible to do so when there is a source of randomness. Many such randomized algorithms exist [AH90, AW92, CIL94, AC08].

Tight $\Theta(n^2)$ bounds are known for the total step complexity (i.e. work) of these randomized algorithms [AC08]. However, there is a large gap between the best known space lower bound of $\Omega(\sqrt{n})$ registers for n-process randomized wait-free consensus using only registers [FHS98] and the $\Omega(n)$ space complexity of existing algorithms. Closing this gap is a long-standing open problem.

Recently, [GHHW13] showed that it is possible to solve an easier, related problem, leader election, in an obstruction-free manner with only $O(\sqrt{n})$ registers. They also gave a technique to transform any obstruction free algorithm with individual step complexity b into a randomized wait-free algorithm which terminates in $O(n(n+b)\log n)$ steps with high probability.

There is an implementation of unbounded timestamps which uses n registers [Lam74]. [EFR07] gave a $\Omega(\sqrt{n})$ lower bound for the space complexity of this problem. Recently, [HHPW14] improved this lower bound to $\Omega(n)$. Inspired by their work, we prove matching upper and lower bounds on the space complexity of randomized (and obstruction-free) binary consensus in a restricted computational model. Specifically, we consider an asynchronous system with n anonymous processes, which communicate through an m-component multi-writer snapshot. Each process alternately performs SCAN and UPDATE, but does not remember information from one UPDATE to the next. We call algorithms designed for this model *memoryless*.

Many existing randomized algorithms [AH90, AW92, CIL94, AC08] may be modified to run in a non-anonymous version of this model. In this case, the additional identifiers are often only used to reserve particular components of the snapshot for a single process. Similarly, existing bounded space anonymous obstruction-free algorithms [GR05, Bow11] can also be modified to run in this model.

Our lower bound uses a *covering* argument (introduced in [BL93]) combined with a *valency* argument (introduced in [FLP85]). The lower bound is applicable to anonymous *nondeterministic solo terminating* [FHS98] binary consensus algorithms, and shows that any such algorithm in this model uses a multi-writer snapshot with at least n components. Obstruction-free and randomized wait-free algorithms are examples of nondeterministic solo terminating algorithms.

The matching upper bound of n components we show in this model improves the space upper bounds given by the existing bounded space anonymous obstruction-free algorithms [GR05, Bow11], which use $8n+2$ components and $2n$ components, respectively. Since an obstruction-free n-component snapshot object can be implemented from n registers [GR05], we also get an upper bound of n registers for anonymous obstruction-free algorithms. Furthermore, using the randomization techniques of [GHHW13], it is possible to obtain an anonymous wait-free randomized algorithm using only n registers which terminates in $O(n^2 \log n)$ steps with high probability.

2 Related Work

We briefly describe some existing obstruction-free consensus algorithms.

Herlihy et al. [HLM03] noted that it is possible to derandomize the randomized consensus algorithm of [AH90], which uses n unbounded single-writer registers, to obtain an obstruction-free

consensus algorithm. Essentially, the coin toss in [AH90] can be replaced with a deterministic choice. This is the best known space upper bound for obstruction-free consensus algorithms.

Guerraoui and Ruppert [GR05] noted that it is possible to derandomize the anonymous randomized consensus algorithm of [Cha96] (more precisely, the modification given by [BPSV06]) in the same way. The result is an obstruction-free consensus algorithm using an unbounded number of ternary registers. They also showed how to modify this algorithm to use only a multi-writer snapshot with $8n + 2$ unbounded components. The resulting algorithm can be modified to be memoryless.

Bowman [Bow11] gave a memoryless anonymous obstruction-free consensus algorithm which uses a multi-writer snapshot with $2n$ ternary components. We note that it is possible to modify their algorithm to use $2n - 2$ components if processes are allowed to remember 1 bit in between UPDATE's, or to use $2n - 1$ components in our memoryless model.

Bouzid et al. [BST11] gave an anonymous obstruction-free consensus algorithm in which each process performs only $O(\sqrt{n})$ writes in every solo execution. This algorithm can be modified to be memoryless, but it uses an unbounded number of registers.

All of the aforementioned algorithms are *race* algorithms in which values 0 and 1 compete to complete *laps*. The first value to get a substantial lead on the other value gets decided. The mechanism for completing a lap varies. If a process gets many consecutive steps, then it continues to compete for the same value and, given enough steps, will complete a lap for that value. Contention may cause a process to switch between the values it is competing for.

3 Model

We consider an asynchronous system with n anonymous processes, which communicate through an m-component multi-writer snapshot S. Each process alternately performs SCAN and UPDATE, but does not remember information from one UPDATE to the next. We call algorithms designed for this system *memoryless*. We do not impose any restrictions on the size of each component of S. In particular, they may be unbounded.

Algorithm 1 is a generic anonymous consensus algorithm in this system. Here, loc_0, loc_1, and loc (resp. val_0, val_1, and val) are functions which, given the result of a SCAN, specify the component of the snapshot object to UPDATE (resp. its new value). The function dec takes the result of a SCAN, returning $\perp$ if a decision cannot yet be reached, and returning the decision value, if a decision can be reached.

Algorithm 1 Generic memoryless anonymous algorithm

```
procedure PROPOSE(x)
    view ← S.scan()
    S.update(loc_x(view), val_x(view))
    loop
        view ← S.scan()
        v ← dec(view)
        if v ≠ ⊥ then
            decide v and terminate
        else
            S.update(loc(view), val(view))
```

An *execution* from a *configuration* C_0 in this system is an alternating sequence of *configurations* and *events*, say C_0, e_1, C_1, e_2, C_2, ..., starting from C_0. Each event e_i (or *step*) is either an UPDATE or a SCAN on S and each configuration C_i consists of the contents of S and the state of each process after event e_i is applied to configuration C_{i-1}. Given a finite sequence of events α and a configuration C, we use $C\alpha$ to denote the configuration obtained from applying the sequence of steps in α starting from C.

We say that a configuration C is *solo v-deciding for process p* if there is a terminating p-only execution starting from C that decides v. We say that C is *solo deciding for p* if it is solo v-deciding for p, for some value v. Finally, we say a consensus algorithm in our model is *nondeterministic solo terminating* if every configuration is solo deciding for every process p.

Two configurations C and C' are *indistinguishable*, denoted $C \sim C'$, to a set of processes P if every process in P has the same state in C as it does in C' and the contents of S are the same in both configurations.

4 Lower Bound

Consider any n-process memoryless anonymous obstruction-free consensus algorithm $\mathcal{A}$ which uses only a multi-writer snapshot S, where $n \geq 2$. We will prove that S has at least n components. Central to the proof are the notions of a *free* process and a (P, q, v)-*bivalent* configuration.

A process is *free* in a configuration C if it has already taken its first SCAN and is poised to perform another SCAN in C. Since processes are anonymous and memoryless, any two free processes in C which see the same SCAN will behave identically. In particular, any solo execution from C of one free process in C is a solo execution from C of any other free process in C, so we can always arrange for two free processes which see the same SCAN to perform identical UPDATEs. We say a configuration C is solo v-deciding *for free processes* if it is solo v-deciding for some free process in C (equivalently, all free processes in C).

A (P, q, v)-*bivalent* configuration, C, is a special kind of bivalent configuration. In C, there is a set P of processes covering distinct components of S and another process $q \notin P$ which covers a component of S (not necessarily distinct from the ones covered by P) such that $C\beta_P(C)$ is solo v-deciding for free processes, and $C\beta_q(C)$ is solo $\overline{v}$-deciding for free processes. We note that every reachable (P, q, v)-bivalent configuration gives a lower bound on the number of components of S. In particular, S must have at least $|P|$ components (in fact, by Lemma 1, $|P| + 1$ components).

Our strategy is to show that, from any $(\{p\}, q, v)$-bivalent configuration C, and any set of processes, Z, that are free in C, it is possible to reach a $(Z \cup \{p\}, q, v')$-bivalent configuration by an execution in which each process in $Z \cup \{p, q\}$ takes at least one step. The proof is by induction on $|Z|$. Suppose $Z' \subsetneq Z$ with $|Z'| = |Z| - 1$. The key step is to show that, for any $(Z' \cup \{p\}, q, v)$-bivalent configuration, it is possible to reach a $(\{p\}, q, v')$-bivalent configuration C' in which at least $|Z'| + 2$ components have been UPDATEd and Z' is free again. By induction, it is possible to reach a $(Z' \cup \{p\}, q, v'')$-bivalent configuration C'' from C'. Among the components UPDATEd starting from C, there is at least one component j which is not covered by $Z' \cup \{p\}$ in C''. Let $z' \in Z' \cup \{p, q\}$ be the last process to UPDATE component j, and let σ' be the scan by z' just before this UPDATE. Now modify the execution from C to C'' to let the remaining free process in $Z - Z'$ perform its SCAN immediately after σ'. This execution applied to C results in a $(Z \cup \{p\}, q, v'')$-bivalent configuration.

We now proceed with the formal proof.

Lemma 1. *Suppose that C is a (P, q, v)-bivalent configuration such that $C\beta_q(C)\beta_P(C)$ is solo v-deciding. Then there is a (P, q, v)-bivalent configuration C' reachable from C by a non-empty q-only execution such that $\beta_q(C')$ is an update by q to a component of S not covered by P and $C'\beta_q(C')\beta_P(C')$ is solo $\overline{v}$-deciding for free processes.*

Proof. Let α be sequence of steps of the longest prefix of q's $\overline{v}$-deciding terminating solo execution from C such that $C\alpha\beta_P(C)$ is solo v-deciding for free processes. Since $C\beta_q(C)\beta_P(C)$ is solo v-deciding for free processes, α is non-empty. Let δ be the next step by q. If δ is a scan or an update to a component of S covered by P, then $C\alpha\delta\beta_P(C) \sim C\alpha\beta_P(C)$ for every free process, so $C\alpha\delta\beta_P$ is solo v-deciding for free processes, which contradicts the definition of α. It follows that δ is an update to a component of S not covered by P. Furthermore, since $C\alpha\delta$ is not solo v-deciding for free processes, it must be solo $\overline{v}$-deciding for free processes. It follows that $C' = C\alpha$ is the desired (P, q, v)-bivalent configuration. □

Lemma 2. *Suppose that C is a $(\{p\}, q, v)$-bivalent configuration. Then there is a $(\{p\}, q, v)$-bivalent configuration D reachable from C by a $\{p, q\}$-only execution in which p and q both take at least two steps.*

Proof. By Lemma 1, we may assume that $C\beta_q(C)\beta_p(C)$ is solo $\overline{v}$-deciding for free processes and $\beta_q(C)$ is an update to a component of S not covered by p. If $C\beta_p(C)\beta_q(C)$ is solo v-deciding for free processes as well, then we can let p, respectively q, take the SCAN which it would take in its v-deciding, respectively $\overline{v}$-deciding, solo terminating execution from $C\beta_p(C)\beta_q(C)$. The resulting configuration D will be a $(\{p\}, q, v)$-bivalent configuration and p and q have both taken two steps to get to D from C. Otherwise, suppose $C\beta_q(C)\beta_p(C)$ is not solo v-deciding for free processes. Since C is a $(\{q\}, p, \overline{v})$-bivalent configuration in which $C\beta_p(C)\beta_q(C) \sim C\beta_q(C)\beta_p(C)$ is not solo v-deciding for free processes, by Lemma 1, there is a $(\{q\}, p, \overline{v})$-bivalent configuration C' reachable from C by a non-empty p-only execution such that $C'\beta_p(C')\beta_q(C')$ is solo v-deciding for free processes. We can now apply the same argument as before on C', reversing the roles of p and q, to obtain D. □

We remark that Lemmas 1 and 2 can be generalized to show that, in any nondeterministic solo terminating consensus algorithm for $n \geq 2$ processes, there is an infinite execution involving only two processes where each process takes infinitely many steps yet no decision is made.

Lemma 3. *Suppose that C is a $(\{p\}, q, v)$-bivalent configuration with a set Z of free processes. Then there is a $(Z \cup \{p\}, q, v')$-bivalent configuration D which is reachable from C by a $(\{p, q\} \cup Z)$-only execution in which each process in $\{p, q\} \cup Z$ takes at least one step.*

Proof. By induction on $|Z|$. The base case is when Z is empty and is taken care of by Lemma 2. Suppose now that Z is non-empty. Fix any process $z \in Z$. By applying the induction hypothesis on C and $Z' = Z \setminus \{z\}$, there is a $(Z' \cup \{p\}, q, u)$-bivalent configuration C' reachable from C by a $(\{p, q\} \cup Z')$-only execution in which each process in $\{p, q\} \cup Z'$ takes at least one step.

By Lemma 1, we may assume that $\beta_q(C')$ is an update to a component of S which is not covered by $\{p\} \cup Z'$ and $C'\beta_q(C')\beta_{Z'\cup\{p\}}(C')$ is solo $\overline{u}$-deciding for free processes. Since $C'\beta_{Z'\cup\{p\}}(C')$ is solo u-deciding for free processes and $C'\beta_{Z'\cup\{p\}}(C')\beta_q(C') \sim C'\beta_q(C')\beta_{Z'\cup\{p\}}(C')$ is solo $\overline{u}$-deciding for free processes, p must perform at least one update in its solo terminating execution from $C'\beta_{Z'\cup\{p\}}(C')$.

Run p from $C'\beta_{Z'\cup\{p\}}(C')$ until it is poised to perform its first UPDATE and call the resulting $(\{p\}, q, u)$-bivalent configuration C''. By applying the induction hypothesis on C'' and Z', we

obtain a $(Z' \cup \{p\}, q, v')$-bivalent configuration C''' which is reachable from C'' by a $(\{p, q\} \cup Z')$-only execution in which each process in $\{p, q\} \cup Z'$ takes at least one step. In particular, q has performed its UPDATE $\beta_q(C')$ so, together with the block UPDATE by $Z' \cup \{p\}$, at least $|Z'| + 2$ distinct components have been UPDATEd by $Z' \cup \{p, q\}$ since C'.

Among these components, there is at least one component j which is not covered by $Z' \cup \{p\}$. Let $z' \in Z' \cup \{p, q\}$ be the last process to UPDATE component j. Consider the last SCAN σ' by z' before its last UPDATE to component j. Since this UPDATE occurs in between C' and C''' and each process in $Z' \cup \{p, q\}$ took at least one step between C and C', σ' must have occurred in between C and C''. If $z' \in \{p, q\}$, then σ' was not the first SCAN by z', since both p and q were both poised to perform updates in C. Otherwise, if $z' \in Z'$, then σ' was not the first SCAN by z', since z' was free in C.

Since σ' occurs in between C and C''' and z was free in C, we can take D to be the configuration which is like C''' except z took its SCAN immediately after σ'. Note that $D \sim C'''$ to every process except z. Since σ' was not the first SCAN by z', we can arrange z to be poised to perform the same UPDATE as z'. Since this UPDATE was the last update to component j in reaching C''', $D\beta_{Z \cup \{p\}}(D) \sim C'''\beta_{Z' \cup \{p\}}(C''')$ to every free process. It follows that D is a $(Z \cup \{p\}, q, v')$-bivalent configuration. □

Theorem 1. *If $\mathcal{A}$ is an n-process memoryless anonymous non-deterministic solo terminating consensus algorithm, then the multi-writer snapshot used by $\mathcal{A}$ has at least n components.*

Proof. Let p_0 start with 0, and let $p_1, p_2, \ldots, p_{n-1}$ start with 1. Call this configuration C_0. By running $p_1, p_2, \ldots, p_{n-1}$ all in lockstep, we may obtain a $(\{p_0\}, p_1, 0)$-bivalent configuration C_0' where $p_1, \ldots, p_{n-1}$ are all poised to perform the same UPDATE β_{p_1} and p_0 is poised to perform an UPDATE β_{p_0}. By applying Lemma 1 and running $p_2, \ldots, p_{n-1}$ in lockstep with p_1 up to, but not including, when p_1 takes its second SCAN, we can obtain a configuration C where $p_2, \ldots, p_{n-1}$ are all free. By Lemma 2, there is a $(\{p_1, \ldots, p_{n-1}\}, p_0, u)$-bivalent configuration D reachable from C. By applying Lemma 1 to D, we obtain a $(\{p_1, \ldots, p_{n-1}\}, p_0, u)$-bivalent configuration D' in which $\beta_{p_0}(D')$ is an update to a component of S not covered by $\{p_1, \ldots, p_{n-1}\}$. Since D' is a configuration in which n components are covered, the multi-writer snapshot used by $\mathcal{A}$ contains at least n components. □

5 Matching Upper Bound

We present an n-process memoryless anonymous obstruction-free binary consensus algorithm, which uses only an n-component multi-writer snapshot object S (Algorithm 2), matching the lower bound we proved earlier. The algorithm can be easily modified to support arbitrary input values. It is possible to view our algorithm as an anonymization of the derandomization of [AH90]. Like that algorithm, ours uses n components (or registers). There is no known randomized or obstruction-free consensus algorithm that uses less space.

Intuitively, values 0 and 1 are competing to complete *laps*. If v gets a substantial lead on $\overline{v}$, then the value v is decided. Initially, each component of the snapshot contains $(0, 0)$. If a process sees this initial state of the snapshot object as a result of a SCAN, then it updates component 1 with $(1, 0)$, if its input is 0, and $(0, 1)$, if its input is 1. Otherwise, it determines the laps, ℓ_0 and ℓ_1, that 0 and 1 are on by finding the largest values in the first and second entries of the components. Then it UPDATEs the smallest component that did not contain (ℓ_0, ℓ_1) with this pair. Now suppose all

the components were equal to (ℓ_0, ℓ_1). If they differ by at least 2, then the process decides the value that is ahead. If $\ell_1 = \ell_0 + 1$, then the process UPDATEs component 1 with $(\ell_0, \ell_1 + 1)$. Otherwise, it UPDATEs component 1 with $(\ell_0 + 1, \ell_1)$. In the first case, it considers 1 to have completed lap ℓ_1. In the second case, it considers 0 to have completed lap ℓ_0.

Algorithm 2 A memoryless anonymous algorithm for n processes using a n-component snapshot object S. Initially, each component is set to $(0, 0)$

```
1:  procedure PROPOSE(x)
2:      loop
3:          [(ℓ_0^(1), ℓ_1^(1)), ..., (ℓ_0^(n), ℓ_1^(n))] ← S.scan()
4:          (ℓ_0, ℓ_1) ← (max_{1≤j≤n} ℓ_0^(j), max_{1≤j≤n} ℓ_1^(j))
5:          if ℓ_0 = ℓ_1 = 0 then                                  ▷ first scan
6:              ℓ_x ← 1
7:          v ← argmax_{v∈{0,1}} ℓ_v                    ▷ break ties in favour of 0
8:          if (ℓ_0^(j), ℓ_1^(j)) = (ℓ_0, ℓ_1) for all j then
9:              if ℓ_v − ℓ_v̄ ≥ 2 then                          ▷ 2 laps ahead already
10:                 decide v and terminate
11:             ℓ_v ← ℓ_v + 1                          ▷ team v is now on next lap
12:         S.update(min{j : (ℓ_0^(j), ℓ_1^(j)) ≠ (ℓ_0, ℓ_1)}, (ℓ_0, ℓ_1))
```

There is a key property that makes this work: After a process p considers a value v to have completed lap ℓ, every process will think that v is on lap at least ℓ after they perform their next SCAN. Moreover, if this causes p to decide v, then every process thinks that $\overline{v}$ is on lap at most $\ell - 1$. This is because there are at most $n - 1$ processes which have out of date information. In the worst case, these processes can overwrite $n - 1$ components of S, but the remaining component of S contains the correct information. Therefore, when a process decides v, every process will see that v is ahead and thus never decide $\overline{v}$.

We now proceed with the formal proof of correctness.

Fix an execution of Algorithm 2. For every SCAN S returning $[(\ell_0^{(1)}, \ell_1^{(1)}), \ldots, (\ell_0^{(n)}, \ell_1^{(n)})]$, let $\ell_0(S) = \max_{1 \le j \le n} \ell_0^{(j)}$, $\ell_1(S) = \max_{1 \le j \le n} \ell_1^{(j)}$, and let $v(S) = \operatorname{argmax}_{v \in \{0,1\}} \ell_v(S)$, breaking ties in favour of 0. Thus, $\ell_0(S)$, $\ell_1(S)$, and $v(S)$ are the values of the scanning process' local variables ℓ_0, ℓ_1, and v by the time line 9 of the code is reached after the scan S.

Lemma 4. *If a scan S by a process p returns a vector which has each component set to $(\ell_0(S), \ell_1(S))$ then, for every scan T taken by some process q no earlier than S, $(\ell_0(T), \ell_1(T)) \ge (\ell_0(S), \ell_1(S))$.*

Proof. By induction. Certainly the claim holds for S. Suppose the claim holds for every scan T' taken no earlier than S and before T. Let T' be the last scan before T. Since processes alternate SCANs and UPDATEs, there are at most n UPDATEs between T' and T. Among the updates between T' and T, there are at most $n - 1$ updates by processes whose last scan was taken before S (since p took the scan S). The updates by processes whose last scan occurs no earlier than S, write integers of the form $(\ell_0', \ell_1') \ge (\ell_0(S), \ell_1(S))$, by the induction hypothesis. It follows that at least one component of S is not updated by a process whose last scan was taken before S. It follows that this component contains either the value $(\ell_0(S), \ell_1(S))$ (if no updates occurred to it between S and T) or $(\ell_0', \ell_1') \ge (\ell_0(S), \ell_1(T))$ (if it was updated by a process whose last scan occurred no

earlier than S). Thus, the vector returned by T contains a pair of integers $(\ell'_0, \ell'_1) \geq (\ell_0(S), \ell_1(S))$, so $(\ell_0(T), \ell_1(T)) \geq (\ell'_0, \ell'_1) \geq (\ell_0(S), \ell_1(S))$. □

Lemma 5. *Let S be a scan by a process p. For every process q, if T is the last scan by q before S, then the next update by q after T, if any, writes integers $(\ell'_0, \ell'_1) \leq (\ell_0(S) + 1, \ell_1(S) + 1)$.*

Proof. We first show that $(\ell_0(T), \ell_1(T)) \leq (\ell_0(S) + 1, \ell_1(S) + 1)$. Consider $\ell_0(T)$. The argument for $\ell_1(T)$ is similar. If $\ell_0(T) = 0$, then certainly $\ell_0(T) \leq \ell_0(S) + 1$ since $\ell_0(S) \geq 0$. Suppose now that $\ell_0(T) > 0$. By considering the first process who wrote $(\ell_0(T), \ell'_1)$ for some $\ell'_1 < \ell_1(T)$, it follows that some process r saw a scan T' before T which returned a vector which had each component set to $(\ell_0(T) - 1, \ell'_1)$. By Lemma 4, since S was taken after T, which was taken after T', $\ell_0(S) \geq \ell_0(T) - 1$, so $\ell_0(T) \leq \ell_0(S) + 1$. Now, if T returned a vector in which each component was set to $(\ell_0(T), \ell_1(T))$, then by Lemma 4, $(\ell_0(S), \ell_1(S)) \geq (\ell_0(T), \ell_1(T))$ and q's next update after T, if any, writes $(\ell'_0, \ell'_1) \leq (\ell_0(S) + 1, \ell_1(S) + 1)$ since $\ell'_{v(T)} = \ell_{v(T)}(T) + 1 \leq \ell_{v(T)}(S) + 1$ and $\ell'_{\overline{v(T)}} = \ell_{\overline{v(T)}}(T) \leq \ell_{\overline{v(T)}}(S)$. Otherwise, q's next update after T, if any, writes $(\ell_0(T), \ell_1(T)) \leq (\ell_0(S) + 1, \ell_1(S) + 1)$. □

Lemma 6. *No two processes decide differently.*

Proof. Let p be the first process to decide some value v and let S be the last scan taken by p. By line 9, S returned a vector which had each component set to $(\ell_0(S), \ell_1(S))$ and, by line 10, $\ell_v(S) + 1 > \ell_{\overline{v}}(S) + 2$, so $\ell_v(S) \geq \ell_{\overline{v}}(S) + 2$. By Lemma 5, for every process q, if T is the last scan by q before S, then q's next update after T, if any, writes integers $(\ell'_0, \ell'_1) \leq (\ell_0(S) + 1, \ell_1(S) + 1)$. Thus, immediately after S, there are at most $n - 1$ processes poised to write integers (ℓ'_0, ℓ'_1) with $\ell'_{\overline{v}} \leq \ell_{\overline{v}}(S) + 1$. By the same argument as in Lemma 4 then, for every scan T taken after S, $\ell_v(T) > \ell_{\overline{v}}(T)$, thus $v(T) = v$. Since processes always decide $v(T)$, where T is the last scan they took, it follows that no process ever decides $\overline{v}$. □

Lemma 7. *If every process proposes v, then no process decides $\overline{v}$.*

Proof. Suppose not. Without loss of generality, suppose $\overline{v} = 0$. Then some process saw a scan T where $v(T) = 0$. Consider the first such scan T. If $\ell_0(T) = 0$, then $\ell_1(T) = 0$ so the process which took T proposed 1, which is impossible. Thus $\ell_0(T) > 0$. By considering the first process who wrote $(\ell_0(T), \ell'_1)$ for some $\ell'_1 < \ell_0(T)$, it follows that some process r saw a scan T' before T which returned a vector which had each component set to $(\ell_0(T) - 1, \ell'_1)$ and had $v(T') = 0$, which contradicts the definition of T. □

Lemma 8. *Every process decides after at most $6n + 1$ consecutive steps.*

Proof. After a process performs an UPDATE, it performs at most n alternating SCANs and UPDATEs to fill S with $(\ell_0(T) + k, \ell_1(T))$, for each $k \in \{0, 1, 2\}$, where T is first SCAN that it sees after its update. After these $6n + 1$ steps, it decides $v(T)$. □

Theorem 2. *There is an n-process memoryless anonymous algorithm for solving obstruction-free consensus which uses only an n-component multi-writer snapshot. Thus, there is an n-process anonymous algorithm for solving obstruction-free consensus which uses only n registers, and a randomized n-process anonymous algorithm for solving wait-free consensus which uses only n registers such that each process terminates with high probability after $O(n^2 \log n)$ steps.*

Proof. Lemmas 6, 7, and 8 show that Algorithm 2 satisfies agreement, validity, and obstruction-free termination, respectively. Since there is an obstruction-free implementation of an n-component multi-writer snapshot from n registers [GR05], the n-component multi-writer snapshot object S may be replaced with n registers. Using the randomization techniques of [GHHW13] and Lemma 8, we may then obtain an anonymous wait-free algorithm which uses only n registers such that each process terminates with high probability after $O(n^2 \log n)$ steps. □

6 Conclusion

We have shown that the space complexity of randomized (and obstruction-free) consensus for n memoryless anonymous processes is exactly n. It is natural to ask what happens if processes are allowed to remember some bits of local memory or if they have identifiers.

We have started to look at the case when processes are only allowed to write 0 or 1. In this case, we are able to show that 3 components are needed for any 2-process obstruction-free consensus algorithm for our model (and this is tight). On the other hand, if each process is allowed 1 bit of local memory, then it is possible to get away with 2 components. Thus, it is conceivable that having some local memory is helpful. (Or, it may only be helpful when the values of the updates are restricted.)

We hope to address these questions in a follow-up paper.

7 Acknowledgements

I would like to thank my advisor, Dr. Faith Ellen. She sparked my interest in this problem and her energy, patience, and insights in discussing this problem and reading many drafts of this work have proven invaluable to me. I would also like to thank David Solymosi for helpful discussions.

References

[AC08] Hagit Attiya and Keren Censor. Tight bounds for asynchronous randomized consensus. *Journal of the ACM (JACM)*, 55(5):20, 2008.

[AH90] James Aspnes and Maurice Herlihy. Fast randomized consensus using shared memory. *Journal of algorithms*, 11(3):441–461, 1990.

[AW92] James Aspnes and Orli Waarts. Randomized consensus in expected $O(n \log^2 n)$ operations per processor. In *Foundations of Computer Science, 1992. Proceedings., 33rd Annual Symposium on*, pages 137–146. IEEE, 1992.

[BL93] James E Burns and Nancy A Lynch. Bounds on shared memory for mutual exclusion. *Information and Computation*, 107(2):171–184, 1993.

[Bow11] Jack R. Bowman. Obstruction-free snapshot, obstruction-free consensus, and fetch-and-add modulo k. Technical Report TR2011-681, Dartmouth College, Computer Science, Hanover, NH, June 2011.

[BPSV06] Harry Buhrman, Alessandro Panconesi, Riccardo Silvestri, and Paul Vitanyi. On the importance of having an identity or, is consensus really universal? *Distributed Computing*, 18(3):167–176, 2006.

[BST11] Zohir Bouzid, Pierre Sutra, and Corentin Travers. Anonymous agreement: The janus algorithm. In *Principles of Distributed Systems*, pages 175–190. Springer, 2011.

[Cha96] Tushar Deepak Chandra. Polylog randomized wait-free consensus. In *Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing*, pages 166–175. ACM, 1996.

[CIL94] Benny Chor, Amos Israeli, and Ming Li. Wait-free consensus using asynchronous hardware. *SIAM Journal on Computing*, 23(4):701–712, 1994.

[EFR07] Faith Ellen, Panagiota Fatourou, and Eric Ruppert. The space complexity of unbounded timestamps. In *Distributed Computing*, pages 223–237. Springer, 2007.

[FHS98] Faith Fich, Maurice Herlihy, and Nir Shavit. On the space complexity of randomized synchronization. *Journal of the ACM (JACM)*, 45(5):843–862, 1998.

[FLP85] Michael J Fischer, Nancy A Lynch, and Michael S Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(2):374–382, 1985.

[GHHW13] George Giakkoupis, Maryam Helmi, Lisa Higham, and Philipp Woelfel. An $O(\sqrt{n})$ space bound for obstruction-free leader election. In *Distributed Computing*, pages 46–60. Springer, 2013.

[GR05] Rachid Guerraoui and Eric Ruppert. What can be implemented anonymously? In *Distributed Computing*, pages 244–259. Springer, 2005.

[HHPW14] Maryam Helmi, Lisa Higham, Eduardo Pacheco, and Philipp Woelfel. The space complexity of long-lived and one-shot timestamp implementations. *Journal of the ACM (JACM)*, 61(1):7, 2014.

[HLM03] Maurice Herlihy, Victor Luchangco, and Mark Moir. Obstruction-free synchronization: Double-ended queues as an example. In *Distributed Computing Systems, 2003. Proceedings. 23rd International Conference on*, pages 522–529. IEEE, 2003.

[Lam74] Leslie Lamport. A new solution of dijkstra's concurrent programming problem. *Communications of the ACM*, 17(8):453–455, 1974.