Anonymous obstruction-free (n,k)-set agreement with n–k+1 atomic read/write registers

Zohir Bouzid, Michel Raynal, Pierre Sutra

Anonymous Obstruction-free (n, k)-Set Agreement with $n - k + 1$ Atomic Read/Write Registers*

Zohir Bouzid†, Michel Raynal⋆,†, Pierre Sutra‡

† IRISA, Université de Rennes, 35042 Rennes, France
⋆ Institut Universitaire de France
‡ Telecom SudParis and CNRS (Université Paris-Saclay)
zohir.bouzid@gmail.com raynal@irisa.fr pierre.sutra@telecom-sudparis.eu

Abstract

The k-set agreement problem is a generalization of the consensus problem. Namely, assuming that each process proposes a value, every non-faulty process must decide one of the proposed values, under the constraint that at most k different values are decided. This is a hard problem in the sense that it cannot be solved in a pure read/write asynchronous system, in which k or more processes may crash. One way to sidestep this impossibility result consists in weakening the termination property, requiring only that a process decides a value if it executes alone during a long enough period of time. This is the well-known *obstruction-freedom* progress condition.

Consider a system of n *anonymous asynchronous* processes that communicate only through atomic *read/write registers*, and such that *any number of them may crash*. This paper addresses and solves the challenging open problem of designing an obstruction-free k-set agreement algorithm with only $(n - k + 1)$ atomic registers. From a shared memory cost point of view, our algorithm is the best algorithm known to date, thereby establishing a new upper bound on the number of registers needed to solve this problem. For the consensus case $(k = 1)$, the proposed algorithm is up to an additive factor of 1 close to the best known lower bound. The paper extends then this algorithm to obtain an x-obstruction-free solution to the k-set agreement problem that employs $(n-k+x)$ atomic registers (with $1 \leq x \leq k < n$), as well as a space-optimal solution for the repeated version of k-set agreement. Using this last extension, we prove that n registers are enough for every colorless task that is obstruction-free solvable with identifiers and any number of registers.

Keywords Anonymous processes, Asynchronous system, Atomic read/write register, Bounded number of registers, Consensus, Distributed algorithm, Distributed computability, Fault-tolerance, k-Set agreement, Obstruction-freedom, Process crash, Repeated k-set agreement, Upper bound, Colorless task.

1 Introduction

Two challenging adversaries: process crashes and anonymity Due to failures, concurrent processes have to deal not only with finite asynchrony (i.e., finite but arbitrary process speed), but also with infinite asynchrony (i.e., process crashes). In this context, mutex-based synchronization becomes inoperative, and pioneering works in *fault-tolerant* distributed computing, e.g., [27, 31], have instead promoted the design of "concurrent reading while writing" algorithms [25, 32, 37].

*A preliminary version of parts of this article appeared in [9].

This new approach to synchronizing concurrent accesses has given rise to novel progress conditions, namely *wait-freedom*, *non-blocking*, and *obstruction-freedom*. Given a concurrent object O, wait-freedom means that any operation on O must terminate if the invoking process does not crash [22]. "Non-blocking", also named *lock-freedom*, states that at least one process that does not crash return from all its operations on O [26]. Obstruction-freedom means that a process returns from its operation on O if it executes solo during a long enough period of time [23].

On the other hand, *anonymous* systems are characterized by the fact that processes have no identity, execute the exact same code and the same initialization of their local variables. Hence, they are – in a strong sense – identical [3, 39], and may only differ from their local input values.

Considering the previous adversaries that are process crashes and anonymity, a fundamental question is the following: "Given a concurrent object, is it possible to implement this object despite the failures of processes and their anonymity?" If the answer is "yes", we are interested in doing it with a small number of multi-writer/multi-reader (MWMR) read/write register.[1]This fundamental question is addressed in this paper, where the concurrent object is k-set agreement, and the progress condition under consideration is obstruction-freedom.

Consensus and k-Set agreement k-Set agreement, introduced in [12], is a generalization of consensus, which corresponds to the case where $k = 1$. In the following, we use the notation (n, k)-set agreement to make explicit the fact that we consider a system of n processes. Every participating process proposes a value, and must decide a value if it does not crash (termination). On the safety side, a decided value must be a value that was proposed by some process (validity), and at most k different values can be decided (agreement).

Impossibility results and the case of obstruction-freedom Designing a deterministic consensus algorithm in a non-anonymous asynchronous system prone to even a single process crash is not possible, be the communication medium a message-passing system [17], or read/write registers [29]. More generally, if k or more processes may crash, there is no deterministic read/write algorithm able to solve (n, k)-set agreement [7, 24, 35]. These impossibility results remain true in anonymous systems.

As we are interested in the computing power of *pure read/write* asynchronous crash-prone *anonymous* systems, we neither want to enrich the underlying system with additional power (e.g., synchrony assumptions, random numbers, or failure detectors [6]), nor impose constraints on the input vector collectively proposed by the processes [19]. This is the reason why, to sidestep the above impossibility results, instead of wait-freedom or non-blocking, we consider the *obstruction-freedom* progress condition [23]. For (n, k)-set agreement, this property states that a process decides a value only if it executes solo during a "long enough" period of time without interruption. The notion of x-obstruction-freedom [38] generalizes this idea to any group of at most x processes. The practical interest of obstruction-freedom is discussed in [16]. An in-depth study of complexity issues of obstruction-free algorithms is presented in [4].

Contributions of the paper: historical perspective This paper presents an *obstruction-free* algorithm solving the (n, k)-set agreement problem in an *asynchronous anonymous read/write* system where any number of processes may crash. Our algorithm makes use of $(n - k + 1)$ MWMR atomic registers, i.e., exactly n registers for consensus. For (n, k)-set agreement, the best lower bound known so far [15] is $\Omega(\sqrt{\frac{n}{k} - 2})$. The anonymous obstruction-free (n, k)-set agreement algorithms presented in [13, 15] requires $2(n - k) + 1$ MWMR registers. Hence our algorithm provides a gain of $(n - k)$ registers.

As far as consensus is concerned, an algorithm is sketched in [41], which (as ours) required n MWMR atomic registers. Always in the case of consensus, Gelashvili [20] proved that $n/20$ registers are necessary, and Zhu proved a lower bound of $n - 1$ registers for non-anonymous consensus [42].

[1]Let us observe that single-writer/multi-reader registers are meaningless in anonymous systems. This is due to the fact that, as processes have no identity, it is not possible to link each of them to some specific registers (for which the process would be the only writer).

Hence, the anonymous consensus algorithm we present in this paper is up to an additive factor of 1 close to the best known lower bound for non-anonymous consensus.

In the *repeated* version of the (n, k)-set agreement problem, processes participate in a sequence of (n, k)-set agreement instances. We show that a simple modification of our base construction solves this problem without additional atomic registers. The authors of [15] prove that $(n - k + 1)$ atomic registers are necessary to solve the repeated (n, k)-set agreement problem in an anonymous system. As a consequence, our solution is space optimal.

Contributions of the paper: technical perspective The proposed algorithms are round-based, following the pattern "snapshot; local computation; write", where the snapshot and write operations occur on $(n - k + 1)$ MWMR registers. This pattern is reminiscent of the one named "look; compute; move" found in robot algorithms [18, 36]. Interestingly, in our base solution and in the x-obstruction-free variation, processes do not maintain any local information between successive rounds. In this sense, these two algorithms are *locally memoryless*.

In our base algorithm, each register contains a quadruplet consisting of a round number, two control bits, and a proposed value. Our algorithm exploits a partial order over the quadruplets. The way a process computes a new quadruplet is the key of this algorithm. The variation for the repeated (n, k)-set agreement problem requires each register to store two additional fields, one of them being the sequence number of the corresponding instance, the other one containing the values decided in the previous (n, k)-set agreement instances. In the x-obstruction-free algorithm, the last field of a quadruplet is a set of values, allowing up to x processes to concurrently progress.

All the algorithms we present hereafter employ a locally memoryless variation of the following pattern: A processes executes a sequence of asynchronous rounds, until it sees two "identical" consecutive rounds. When this occurs, the process can decide a value, and no other process will be able to decide a different value in the next rounds. Multiple *wait-free* consensus algorithm were proposed in the past that apply this pattern (see e.g., [5, 14, 32, 33, 34]). These algorithms cover anonymous and non-anonymous systems, binary and multivalued consensus. Some use an eventual leader failure detector Ω [11], while others rely on randomization.

Contributions of the paper: universality perspective In addition to the anonymous algorithms for consensus and k-set agreement, this paper also presents a *universal construction* [22], suited to the obstruction-freedom progress condition, for concurrent objects which can be implemented in an anonymous system with any number of MWMR atomic registers. This universal construction, which is based on the previous repeated consensus algorithm requires only n MWMR registers.

Hence, it follows from this universal construction that, for the obstruction-freedom progress condition, distributed objects implementable anonymously with any number of MWMR atomic registers require in fact n registers only.

Furthermore, in regard to colorless tasks [8] (i.e., distributed problems that do not require some form of symmetry breaking), we show that identifiers do not bring more computational power. A similar result (for colorless tasks) has been recently presented in [40]. While our approach is constructive, the one described in [40] is based on topology.

Roadmap Section 2 presents the computing model and definitions used in this paper. Section 3 depicts a base anonymous obstruction-free algorithm solving consensus. This algorithm captures the essence of our solution. Its correctness is proved in Section 4. Section 5 extends it to solve (n, k)-set agreement. The case where (n, k)-set agreement is used repeatedly is addressed in Section 6. Section 7 considers the x-obstruction-freedom progress condition, and presents a solution using $(n - k + x)$ registers. The reduction results on the power of repeated anonymous consensus (universal construction) are presented in Section 8. Finally, Section 9 closes the paper.

2 Computing Model and Problem Definition

2.1 Computing Model

Process model The distributed system is composed of n asynchronous processes $\{p_1, \ldots, p_n\}$. When considering a process p_i, the integer i is called its *index*. Indexes are used to ease the exposition from an external observer point of view. Processes do not have identities and execute the very same code. It is assumed they know the value n.

Let $\mathbb{T}$ denote the increasing sequence of time instants (observable only from an external point of view). At each instant, a unique process is activated to execute a step. A *step* consists in a read or a write to a register (access to the shared memory) possibly followed by a finite number of internal operations (on local variables).

Up to $(n-1)$ processes may crash. Before crashing a process executes correctly its algorithm. After it crashed (if it ever does), a process executes no more step. From a terminology point of view, and given an execution, a *faulty* process is a process that crashes, and a *correct* process is a process that does not crash. No process knows if it is correct or faulty (this is because, before crashing, a faulty process behaves as a correct one).

Communication model In addition to processes, the computing model includes a communication medium made up of m multi-writer/multi-reader (MWMR) atomic registers (let us notice that anonymity prevents processes from using single-writer/multi-reader registers). Registers are encapsulated in an array denoted $REG[1..m]$. The registers are *atomic*. This means that read and write operations appear as if they have been executed sequentially, and this sequence (a) respects the real-time order of non-concurrent operations, and (b) is such that each read returns the value written by the closest preceding write operation (or the initial value of the register if there is no preceding write operation) [28]. When considering the set of concurrent objects defined from a sequential specification, atomicity is named *linearizability* [26], and the sequence of operations is called a *linearization*. Moreover, the time instant at which an operation appears as being executed is called its *linearization point*.

From atomic registers to a snapshot object At the upper layer (where consensus and (n,k)-set agreement are solved), we use the array $REG[1..m]$ to construct a snapshot object [1]. This object, denoted REG hereafter, provides processes with the operations write() and snapshot(). When a process invokes REG.write(x,v), it deposits the value v in $REG[x]$. When it invokes REG.snapshot() it obtains the content of the whole array. The snapshot object is linearizable, i.e., every invocation of REG.snapshot() appears as instantaneous. For the REG object, a linearization is a sequence of write and snapshot operations.

An anonymous non-blocking (hence obstruction-free) implementation of a snapshot object is described in [21]. This implementation does not require additional atomic registers. In the following, we consider that the snapshot object REG is implemented using this algorithm.

2.2 Obstruction-free consensus and obstruction-free (n,k)-set agreement

Obstruction-free consensus A consensus object is a one-shot object that provides each process with a single operation denoted propose(). "*One-shot*" means that a process invokes propose() at most once. When a process invokes propose(v), we say that it "proposes v". When the invocation of propose() returns value v', we say that the invoking process "decides v'".

A process executes "solo" when it keeps on executing while the other processes have stopped their execution (at any point of their algorithm). In the context of the obstruction-free progress condition (see below the OF-termination property), the consensus problem (consequently called *obstruction-free consensus*) is defined by the following properties (that is, to be correct, any obstruction-free algorithm

must satisfy such properties).

- Validity. If a process decides a value, this value was proposed by a process.
- Agreement. No two processes decide different values.
- OF-termination. If there is a time after which a correct process executes solo, it decides a value.
- SV-termination. If a single value is proposed, all correct processes decide.

Validity relates outputs to inputs. Agreement relates the outputs. Termination states the conditions under which a correct process must decide. There are two cases. The first is related to obstruction-freedom. The second one is independent of the concurrency and failure pattern; it is related to the input value pattern [2].

Obstruction-free (n, k)-set agreement An obstruction-free (n, k)-set agreement object is a one-shot object which has the same validity, OF-termination, and SV-termination properties as consensus, and for which we replace the agreement property with:

- Agreement. At most k different values are decided.

Section 3, which follows, describes a basic algorithm that solves the obstruction-free anonymous consensus problem. This algorithm will be extended in Section 5 to solve (n, k)-set agreement, and in Section 7 to address the x-obstruction-freedom progress condition (instead of obstruction-freedom).

3 Obstruction-free Anonymous Consensus Algorithm

The obstruction-free anonymous consensus algorithm is presented in Figure 2. As indicated in the Introduction, from a data structure point of view, its essence is captured by quadruplets written in the MWMR atomic registers.

Shared memory The shared memory is made up of a snapshot object REG, composed of n MWMR atomic registers. Each of them contains a quadruplet initialized to $\langle 0, \texttt{down}, \texttt{false}, \perp \rangle$. The meaning of these fields is the following.

- The first field, denoted rd, is a round number.
- The second field, denoted $\ell v \ell$ (level), has a value in $\{\texttt{up}, \texttt{down}\}$, where $\texttt{up} > \texttt{down}$.
- The third field, denoted $cf\ell$ (conflict), is a Boolean (initially equals to `false`). We assume $\texttt{true} > \texttt{false}$.
- The last field, denoted $va\ell$ is initialized to $\perp$ (default value that cannot be proposed). It then always contains a proposed value. It is assumed that the set of proposed values is totally ordered, and that $\perp$ is smaller than any proposed value.

When considering lexicographical ordering, it is easy to see that the set of all the possible quadruplets $\langle rd, \ell v \ell, cf\ell, va\ell \rangle$ is totally ordered. This total order, and its reflexive closure, are denoted "$<$" and "$\leq$", respectively.

Notion of conflict and the function sup() The function sup(), defined in Figure 1, plays a central role in the algorithm. It takes a non-empty set of quadruplets T as input parameter, and returns a quadruplet, which is the supremum of T, defined as follows.

Let $\langle r, level, conflict, v \rangle$ be the maximal element of T according to the lexicographical ordering (line S1), and $tuples(T)$ be the set of tuples of T associated with the maximal round number r (line S2). The set T is *conflicting* if either $conflict$ is true or the set $tuples(T)$ contains more than one element (line S3).

[2] In an anonymous system, the input values are the only way to distinguish the processes that have different inputs. It follows that the case where all the processes propose the same value is similar to the case where a single process executes.

```
function sup(T) is                                          % T is a set of quadruplets %
(S1) let ⟨r, levelℓ, conflict, v⟩ be max(T);                % lexicographical order %
(S2) let tuples(T) be {X | X ∈ T ∧ X.rnd = r};
(S3) let conflict(T) be conflict ∨ |tuples(T)| > 1;
(S4) return(⟨r, levelℓ, conflict(T), v⟩).
```

Figure 1: The function sup()

Function $\mathsf{sup}(T)$ first checks whether T is conflicting (lines S2-S3). Then, it returns at line S4 the quadruplet $\langle r, \ell evel, conf\ell ict(T), v\rangle$, where $conf\ell ict(T)$ indicates if the input set T is conflicting. Let us notice that, since $\mathtt{true} > \mathtt{false}$, the quadruplet returned by $\mathsf{sup}(T)$ is always greater than, or equal to, the greatest element in T, i.e., $\mathsf{sup}(T) \geq \mathsf{max}(T)$.

```
operation propose(v_i) is
(01) repeat forever
(02)     view ← REG.snapshot();
(03)     case (∃r > 0, val : ∀z : view[z] = ⟨r, up, false, val⟩) then
                 return(val);
(04)         (∃r > 0, val : ∀z : view[z] = ⟨r, down, false, val⟩) then
                 REG.write(1, ⟨r + 1, up, false, val⟩);
(05)         (∃r > 0, val, levelℓ : ∀z : view[z] = ⟨r, levelℓ, true, val⟩) then
                 REG.write(1, ⟨r + 1, down, false, val⟩);
(06)         otherwise let ⟨r, levelℓ, cfℓ, valℓ⟩
                                   ← sup(view[1], ··· , view[n], ⟨1, down, false, v_i⟩);
(07)             z ← smallest index y such that view[y] ≠ ⟨r, levelℓ, cfℓ, valℓ⟩;
(08)             REG.write(z, ⟨r, levelℓ, cfℓ, valℓ⟩);
(09)     end case
(10) end repeat.
```

Figure 2: Anonymous obstruction-free consensus

The algorithm The base construction is pretty simple, and consists in an appropriate management of the snapshot object REG, so that the n quadruplets it contains (a) never allow validity or agreement to be violated, and (b) eventually allow termination under good circumstances (which occur when obstruction-freedom is satisfied or when a single value is proposed).

In Figure 2, when a process p_i invokes $\mathsf{proposes}(v_i)$, it enters a loop that it will exit at line 03 (provided it terminates) with the statement $\mathsf{return}(val\ell)$, where $val\ell$ is the decided value. After entering the loop, a process issues a snapshot and assigns the returned array to its local variable $view[1..n]$ (line 02). Then, there are two main cases according to the value stored in $view$.

- Case 1 (lines 03-05). All entries of $view_i$ contain the same quadruplet $\langle r, \ell evel, conf\ell ict, val\ell\rangle$, and $r > 0$. Then, there are three sub-cases to consider.
  - Case 1.1. If the level is up and the conflict is false, the invoking process decides the value $val\ell$ (line 03).
  - Case 1.2. If the level is down and the conflict field is false, process p_i enters the next round by writing $\langle r + 1, \mathtt{up}, \mathtt{false}, val\rangle$ in the first entry of REG (line 04).
  - Case 1.3. If there is a conflict, p_i enters the next round by writing $\langle r + 1, \mathtt{down}, \mathtt{false}, val\rangle$ in the first entry of REG (line 05).
- Case 2 (lines 06-08). Not all the entries of $view_i$ are equal, or one of them contains the quadruplet $\langle 0, -, -, -\rangle$. In such a case, p_i first calls $\mathsf{sup}(view[1], \cdots, view[n], \langle 1, \mathtt{down}, \mathtt{false}, v_i\rangle)$ (line 06), which returns a quadruplet X greater than all the input quadruplets, or equal to the

greatest of them. As we have seen previously, this quadruplet X may inherit or discover a conflict. Moreover, as $\langle 1, \texttt{down}, \texttt{false}, v_i\rangle$ is an input parameter of $\texttt{sup}()$, $X.val$ cannot equal $\perp$. As none of the predicates at lines 03-05 is satisfied, at least one entry of $view[1..n]$ is different than X. Then process p_i writes X into $REG[z]$, where, from its point of view, z is the first entry of REG whose content differs from X (lines 07-08).

The underlying operational intuition As indicated in the Introduction, the intuition that underlies this algorithm is similar to the one used in some non-anonymous consensus algorithms (e.g., [5, 32, 34]), namely, if a process executes two consecutive rounds (scans) returning the same values, it can safely decide. To understand it, let us first consider the very simple case where a single process p_i executes the algorithm. From its first invocation of $REG.\texttt{snapshot}()$ (line 02), it obtains a view $view$ in which all the elements are equal to $\langle 0, \texttt{down}, \texttt{false}, \perp\rangle$. Hence, p_i executes line 06, where the invocation of $\texttt{sup}()$ returns the quadruplet $\langle 1, \texttt{down}, \texttt{false}, v_i\rangle$, that is written into $REG[1]$ at line 08. Then, during the second round, p_i computes with the help of function $\texttt{sup}()$ again the quadruplet $\langle 1, \texttt{down}, \texttt{false}, v_i\rangle$, and writes it into $REG[2]$; etc., until p_i writes $\langle 1, \texttt{down}, \texttt{false}, v_i\rangle$ in all the atomic registers of $REG[1..n]$. When this occurs, p_i obtains at line 02 a view where all the elements equal $\langle 1, \texttt{down}, \texttt{false}, v_i\rangle$. It consequently executes line 04 and writes $\langle 2, \texttt{up}, \texttt{false}, v_i\rangle$ in $REG[1]$. Then, during the following rounds, process p_i writes $\langle 2, \texttt{up}, \texttt{false}, v_i\rangle$ in the other registers of REG (line 08). When this is done, p_i obtains a snapshot containing solely $\langle 2, \texttt{up}, \texttt{false}, v_i\rangle$, and when this occurs, p_i executes line 03 where it decides the value v_i.

Let us now consider the case where, while p_i is executing, another process p_j invokes $\texttt{propose}(v_j)$ with $v_j = v_i$. It is easy to see that, in such a case, p_i and p_j collaborate to fill in REG with the same quadruplet $\langle 2, \texttt{up}, \texttt{false}, v_i\rangle$. If $v_j \neq v_i$, depending on the concurrency pattern, a conflict may occur. For instance, it occurs if REG contains both $\langle 1, \texttt{down}, \texttt{false}, v_i\rangle$ and $\langle 1, \texttt{down}, \texttt{false}, v_j\rangle$. If a conflict appears, it will be propagated from round to round, until a process executes alone a higher round.

Remark 1 Let us first notice that no process needs to memorize in its local memory the values that it will use during the next rounds. Not only processes are anonymous, but their code is also memoryless (no persistent variables). The snapshot object REG constitutes the whole memory of the system. Hence, as defined in the Introduction, the algorithm is locally memoryless. In this sense, and from a locality point of view, it has a "functional" flavor.

Remark 2 Let us consider the n-bounded concurrency model [2, 30]. This model is made up of an arbitrary number of processes, but, at any time, there are at most n processes executing steps. This allows processes to leave the system and other processes to join it as long as the concurrency degree does not exceed n.

The previous algorithm works without modification in such a model. A proposed value is now a value proposed by any of the N processes that participate in the algorithm. Hence, if $N > n$, the number of proposed values can be greater than the upper bound n on the concurrency degree. This versatility dimension of the algorithm is a direct consequence of the previous locally memoryless property.

4 Proof of the Algorithm

This section presents a correctness proof of the previous obstruction-free anonymous consensus algorithm. After a few definitions provided in Section 4.1, Section 4.2 shows that a relation "$\sqsupseteq$" defined over the quadruplets is a partial order. This relation is central to prove the key properties of the algorithm. Such properties are established in Sections 4.3 and 4.4. Then, based on these properties, Section 4.5 shows that the algorithm is correct.

4.1 Definitions and notations

Let $\mathcal{E}$ be a set of quadruplets that can be written in REG. Given $X \in \mathcal{E}$, its four fields are denoted $X.rd, X.\ell v\ell, X.cf\ell$ and $X.va\ell$, respectively. Relations $>$ and $\geq$ refer to the lexicographical ordering over $\mathcal{E}$. Moreover, where appropriate, we consider the array $view[1..n]$ as the set $\{view[1], \cdots, view[n]\}$.

Definition 1. *Let $X, Y \in \mathcal{E}$.*

$$X \sqsupset Y \stackrel{def}{=} (X > Y) \wedge [(X.rd > Y.rd) \vee (X.cf\ell)].$$

$X \sqsupset Y$ (X "dominates" Y) if (a) X is lexicographically greater than Y and (b) the round of X is greater than the round of Y or X indicates a conflict (in this case, as $X > Y$, we have $X.rd = Y.rd$).

At the operational level the algorithm ensures that the quadruplets it generates are totally ordered by the relation $>$. Differently, the relation $\sqsupset$ (which is a partial order on these quadruplets, see Section 4.2) captures the relevant part of this total order, and is consequently the key cornerstone on which the proof of the algorithm relies.

When $X \sqsupset Y$ holds, we say "*X strictly dominates Y*". Similarly, "*X dominates Y*", denoted $X \sqsupseteq Y$, means that $(X \sqsupset Y)$ or $(X = Y)$ holds. Relations $\sqsubset$ and $\sqsubseteq$ are defined in the natural way.

Definition 2. *Given a set of quadruplets T, T is* homogeneous *if it contains a single element, say X. We then write it "T is $\mathcal{H}(X)$".*

Notation 1. *The value of the local variable $view$ of a process p_i at time τ, is denoted $view_i^\tau$. Similarly the value of an atomic register $REG[x]$ at time τ is denoted $REG^\tau[x]$, and the value of REG at time τ is denoted REG^τ.*

Notation 2. *$\mathcal{W}(x, X)$ denotes the writing of a quadruplet X in the register $REG[x]$.*

Definition 3. *We say "a process p_j* covers *$REG[x]$ at time τ" when its next non-local step after time τ is $\mathcal{W}(x, X)$, where X is the quadruplet which is written. In this case we also say "$\mathcal{W}(x, X)$* covers *$REG[x]$ at time τ" or "$REG[x]$* is covered *by $\mathcal{W}(x, X)$ at time τ".*

Let us notice that if p_j covers $REG[x]$ at time τ, then τ necessarily lies between the last snapshot issued by p_j at line 02 and its planned write $\mathcal{W}(x, X)$ that will occur at line 04, 05, or 08.

4.2 The relation $\sqsupseteq$ is a partial order

Lemma 1. *$\sqsupseteq$ is a partial order.*

Proof The antisymmetry property of $\sqsupseteq$ follows from the eponymous property of $>$. To prove the transitivity property, let us assume that $X \sqsupseteq Y$ and $Y \sqsupseteq Z$. We have to show that $X \sqsupseteq Z$. If $X = Y$ or $Y = Z$, the claim follows trivially. Hence, assume that $X \sqsupset Y$ and $Y \sqsupset Z$ and let us prove that $X \sqsupset Z$. Observe that, due to the definition of $\sqsupset$, we have $\big((X \sqsupset Y) \wedge (Y \sqsupset Z)\big) \Rightarrow \big((X > Y) \wedge (Y > Z)\big)$. As $(X > Y) \wedge (Y > Z)$, it follows by transitivity of $>$ that $X > Z$. To prove $X \sqsupset Z$, it thus remains to show that $\big((X.rd > Z.rd) \vee (X.cf\ell)\big)$.

Let us observe that, due to the definition of $\sqsupset$, we have $(X \sqsupset Y) \Rightarrow \big((X.rd > Y.rd) \vee (X.cf\ell)\big)$. If $(X.cf\ell)$ then the claim follows trivially. So assume in following that $(X.cf\ell = \texttt{false})$. Therefore, $(X.rd > Y.rd)$. But as $(Y > Z)$, we have $(Y.rd \geq Z.rd)$. By transitivity this yields $(X.rd > Z.rd)$. This, combined with the fact that $X > Z$ showed above, implies that $X \sqsupset Z$. $\square_{Lemma\ 1}$

4.3 Extracting the relations $\sqsupset$ and $\sqsupseteq$ from the algorithm

The definition of the function $\mathsf{sup}()$, which takes a non-empty set of quadruplets as input parameter was given in Figure 1. The next lemma shows that the quadruplet returned by $\mathsf{sup}(T)$, dominates all the elements of T.

Lemma 2. *Let T be a non-empty set of quadruplets. $\forall\, X \in T : \mathsf{sup}(T) \sqsupseteq X$.*

Proof Let $X \in T$ and $S = \mathsf{sup}(T)$. We have to prove that $S \sqsupseteq X$. Let us first observe that, as $S = \mathsf{sup}(T) \geq \mathsf{max}(T) \geq X$, we have $S \geq X$. If $S = X$ then the lemma follows immediately. So let us assume in the following that $S > X$. To prove $S \sqsupset X$, we need to show that $\big((S.cf\ell) \vee (S.rd > X.rd)\big)$. Assume that $(S.cf\ell = \mathtt{false})$ and let us prove that $(S.rd > X.rd)$.

As $(\mathsf{sup}(T).cf\ell = \mathtt{false})$, it follows from the code in Figure 1 that $conf\ell ict(T) = \mathtt{false}$ and $\mathsf{sup}(T) = \mathsf{max}(T)$. Therefore, $\mathsf{sup}(T)$ is the unique quadruplet of T associated with the round number $\mathsf{sup}(T).rd$. All other elements of T if any have a strictly smaller round number. Therefore, $S.rd > X.rd$. This establishes the claim. $\square_{Lemma\ 2}$

Lemma 3. *If p_i executes $\mathcal{W}(-, Y)$ at time τ, then for every $X \in view_i^\tau : Y \sqsupseteq X$.*

Proof We consider two cases according to the line at which the write occurs.

- Y is written at line 04 or 05. It follows that $Y.rd = (\mathsf{max}(view_i^\tau).rd) + 1$. Therefore, for every $X \in view_i^\tau : Y.rd > X.rd$. Hence $Y \sqsupset X$.
- Y is written at line 08. In this case, due to the call to function $\mathsf{sup}()$ at line 06, the value Y written by p_i is equal to $\mathsf{sup}(T)$ where $T = \{view_i^\tau[1], \cdots, view_i^\tau[n], \langle 1, \mathtt{down}, \mathtt{false}, v_i\rangle\}$. According to Lemma 2, it follows that for every $X \in view_i^\tau$ we have $Y = \mathsf{sup}(T) \sqsupseteq X$.

$\square_{Lemma\ 3}$

Lemma 4. *Let us assume that no process is covering $REG[x]$ at time τ. For every write $\mathcal{W}(-, X)$ that (a) occurs after τ and (b) was not covering a register of REG at time τ, we have $X \sqsupseteq REG^\tau[x]$.*

Proof The proof is by contradiction. Let p_i be the first process that executes a write $\mathcal{W}(-, X)$ contradicting the lemma. This means that $\mathcal{W}(-, X)$ is not covering a register of REG at time τ and $X \not\sqsupseteq REG^\tau[x]$. Let this write occur at time $\tau_2 > \tau$. Thus, all writes that take place between τ and τ_2 comply with the lemma. We derive a contradiction by showing that $X \sqsupseteq REG^\tau[x]$.

Let $\tau_1 < \tau_2$ be the linearization time of the last snapshot taken by p_i (line 02) before executing $\mathcal{W}(-, X)$. Since $\mathcal{W}(-, X)$ was not covering a register of REG at time τ, the snapshot preceding this write was necessarily taken after τ. That is, $\tau_1 > \tau$, and we have $\tau_2 > \tau_1 > \tau$.

According to Lemma 3, $X \sqsupseteq view_i^{\tau_2}[x]$. But since the snapshot returning $view_i^{\tau_2}$ is linearized at τ_1, it follows that $view_i^{\tau_2} = REG^{\tau_1}$. Therefore, we have $X \sqsupseteq REG^{\tau_1}[x]$ (let us call this assertion R).

In the following we show that $REG^{\tau_1}[x] \sqsupseteq REG^\tau[x]$. If $REG[x]$ was not updated between τ and τ_1, then $REG^{\tau_1}[x] = REG^\tau[x]$ and the claim follows. Otherwise, if $REG[x]$ was updated between τ and τ_1, the content of $REG^{\tau_1}[x]$, let it be Y, is the result of a write $\mathcal{W}(x, Y)$ that occurred between τ and τ_1 and that was not covering a register of REG at time τ (let us remember that no write is covering $REG[x]$ at time τ, which is crucial in the proof). We assumed above that τ_2 is the first time at which the lemma is contradicted. Hence the write $\mathcal{W}(x, Y)$, which occurs before τ_2, complies with the requirements of the lemma. It follows that $Y \sqsupseteq REG^\tau[x]$, and we consequently have $REG^{\tau_1}[x] \sqsupseteq REG^\tau[x]$.

But it was shown above (see assertion R) that $X \sqsupseteq REG^{\tau_1}[x]$. Hence, due to the transitivity of the relation $\sqsupseteq$ (Lemma 1), we obtain $X \sqsupseteq REG^\tau[x]$, a contradiction that concludes the proof of the lemma.

$\square_{Lemma\ 4}$

Lemma 5. *Let τ and $\tau' \geq \tau$ be two time instants. If $REG^{\tau'}$ is $\mathcal{H}(Y)$, then there exists $X \in REG^{\tau}$ such that $Y \sqsupseteq X$.*

Proof If $REG^{\tau'} = REG^{\tau}$, the lemma holds trivially. So let us assume in the following that $REG^{\tau'} \neq REG^{\tau}$ which means that a write happens between τ and τ'. If $\langle 0, \mathtt{down}, \mathtt{false}, \perp\rangle \in REG^{\tau}$, as every quadruplet Y written in REG is such that $Y.rd \geq 1$ (line 04, 05, or lines 06-08), we have $Y \sqsupset \langle 0, \mathtt{down}, \mathtt{false}, \perp\rangle$.

So, let us assume that $\langle 0, \mathtt{down}, \mathtt{false}, \perp\rangle \notin REG^{\tau}$ and consider the last write in REG before τ. Assume this happens at $\tau^- \leq \tau$ and let p_i be the writing process. Process p_i has no write covering a register of REG at time τ^-. Consequently, at most $(n-1)$ processes[3] have a write covering a register of REG at time τ^-. Hence, there exists $x \in \{1, \ldots, n\}$ such that no write is covering $REG[x]$ at time τ^-. Let $X = REG^{\tau^-}[x] = REG^{\tau}[x]$. If $X = Y$ then the claim of the lemma follows trivially. So assume in the following that $X \neq Y$. Since $REG^{\tau^-}[x] = X$, $REG^{\tau'}[x] = Y$ and $Y \neq X$, there is necessarily a write $\mathcal{W}(x, Y)$ that occurred between τ^- and τ'. As this write was not covering a register of REG at time τ^-, it follows (according to Lemma 4) that $Y \sqsupseteq X$, which proves the lemma. $\square_{Lemma\ 5}$

The two following lemmas are corollaries of Lemma 5.

Lemma 6. *If REG^{τ} is $\mathcal{H}(X)$, $REG^{\tau'}$ is $\mathcal{H}(Y)$, and $\tau' \geq \tau$, then $Y \sqsupseteq X$.*

Lemma 7. *If REG^{τ} is $H(X)$, $REG^{\tau'}$ is $\mathcal{H}(Y)$, $\tau' \geq \tau$, $(Y.rd = X.rd)$ and $(\neg Y.cf\ell)$ then $(Y.va\ell = X.va\ell)$.*

Proof According to Lemma 6, $Y \sqsupseteq X$. If $Y = X$ then the claim follows immediately. So let us assume $Y \sqsupset X$. As $(Y.rd = X.rd)$ and $(\neg Y.cf\ell)$, the definition of $\sqsupseteq$ implies that $Y.va\ell = X.va\ell$. $\square_{Lemma\ 7}$

4.4 Exploiting homogeneous snapshots

Lemma 8. $[(X \in REG^{\tau}) \wedge (X.\ell v\ell = \mathtt{up})] \Rightarrow \big(\exists\ \tau' < \tau$*: $REG^{\tau'}$ is $\mathcal{H}(Z)$, where $Z = \langle X.rd - 1, \mathtt{down}, \mathtt{false}, X.va\ell\rangle\big)$.*

Proof Let us first show that there is a process that writes the quadruplet X' into REG, with $X' = \langle X.rd, X.\ell v\ell, \mathtt{false}, X.va\ell\rangle$. We have two cases depending on the value of $X.cf\ell$.

- If $X.cf\ell = \mathtt{false}$, then let $X' = X$. Since $X.\ell v\ell = X'.\ell v\ell = \mathtt{up}$, X was necessarily written into REG by some process (let us recall that the initial value of each register of REG is $\langle 0, \mathtt{down}, \mathtt{false}, \perp\rangle$).
- If $X.cf\ell = \mathtt{true}$, let us consider the time τ_1 at which X was written for the first time into REG, say by p_i. Since $X.\ell v\ell = \mathtt{up}$, both τ_1 and p_i are well defined. This write of X happens necessarily at line 08 (If it was at line 04 or 05, we would have $X.cf\ell = \mathtt{false}$).

  Therefore, X was computed at line 06 by the function $\mathtt{sup}()$. Namely we have $X = \mathtt{sup}(T)$, where the set T is equal to $\{view^{\tau}[1], \cdots, view^{\tau}[n], \langle 1, \mathtt{down}, \mathtt{false}, v_i\rangle\}$. Observe that $X \notin T$, otherwise X would not be written for the first time at τ_1. Let $X' = \mathtt{max}(T)$. Since $X \notin T$, it follows that $X \neq X'$. Due to line S4 of the function $\mathtt{sup}()$, X and X' differ only in their conflict field. Therefore, as $X.cf\ell = \mathtt{true}$, it follows that $X'.cf\ell = \mathtt{false}$. Finally, as $X'.\ell v\ell = \mathtt{up}$ and all registers of REG are initialized to $\langle 0, \mathtt{down}, \mathtt{false}, \perp\rangle$, it follows that X' was necessarily written into REG by some process.

[3] Let us notice that this is the only place in the proof where the consensus version of the algorithm requires more than $(n-1)$ MWMR atomic registers.

In both cases, there exists a time at which a process writes $X' = \langle X.rd, X.\ell v\ell, \texttt{false}, X.va\ell\rangle$ into REG. Let us consider the first process p_i that does so. This occurs at some time $\tau_2 < \tau$. As $X'.\ell v\ell = \texttt{up}$, this write can occur only at line 04 or line 08.

We first show that this write occurs necessarily at line 04. Assume for contradiction that the write of X' into REG happens at line 08. In this case, the quadruplet X' was computed at line 06. Therefore, $X' = \mathsf{sup}(T)$ where where the set T is equal to $\{view^{\tau_2}[1], \cdots, view^{\tau_2}[n], \langle 1, \texttt{down}, \texttt{false}, v_i\rangle\}$. Observe that $\mathsf{sup}(T)$ and $\mathsf{max}(T)$ can differ only in their conflict field. As $\mathsf{sup}(T).cf\ell = X'.cf\ell = \texttt{false}$, it follows that $X' = \mathsf{sup}(T) = \mathsf{max}(T)$. Consequently, $X' \in view^{\tau_2}$. That is, p_i is not the first process that writes X' in REG, contradiction. Therefore, the write necessarily happens at line 04.

From the precondition at line 04, $view^{\tau_2}$ is $\mathcal{H}(\langle X'.rd - 1, \texttt{down}, \texttt{false}, X'.va\ell\rangle)$, and the lemma holds. $\square_{Lemma\ 8}$

Lemma 9. $[(REG^{\tau}$ *is* $\mathcal{H}(X)) \wedge (X.\ell v\ell = \texttt{up}) \wedge (\neg X.cf\ell) \wedge (REG^{\tau'}$ *is* $\mathcal{H}(Y)) \wedge (Y.rd \geq X.rd)] \Rightarrow (Y.va\ell = X.va\ell)$.

Proof The proof is by induction on $Y.rd$. Let us first assume that $Y.rd = X.rd$, for which we consider two cases.

- Case 1: $\tau \geq \tau'$. Since $X.cf\ell = \texttt{false}$, it follows according to Lemma 7 that $Y.va\ell = X.va\ell$.
- Case 2: $\tau' > \tau$. According to Lemma 6, $Y \sqsupseteq X$. As $Y.rd = X.rd$, it follows that $Y.\ell v\ell \geq X.\ell v\ell = \texttt{up}$, and consequently $Y.\ell v\ell = \texttt{up}$.

  Summarizing we have $REG^{\tau'}$ is $\mathcal{H}(Y)$, $Y.\ell v\ell = \texttt{up}$ and $Y.rd = X.rd$. According to Lemma 8, this implies that there is $\tau_1 < \tau$ and $\tau_1' < \tau'$ such that REG^{τ_1} is $\mathcal{H}(\langle X.rd-1, \texttt{down}, \texttt{false}, X.va\ell\rangle)$ and $REG^{\tau_1'}$ is $\mathcal{H}(\langle Y.rd - 1, \texttt{down}, \texttt{false}, Y.va\ell\rangle)$. According to Lemma 6, we have

  - either $\langle X.rd - 1, \texttt{down}, \texttt{false}, X.va\ell\rangle \sqsupseteq \langle Y.rd - 1, \texttt{down}, \texttt{false}, Y.va\ell\rangle$,
  - or $\langle Y.rd - 1, \texttt{down}, \texttt{false}, Y.va\ell\rangle \sqsupseteq \langle X.rd - 1, \texttt{down}, \texttt{false}, X.va\ell\rangle$.

  Since by assumption $X.rd = Y.rd$, it follows that $X.va\ell = Y.va\ell$.

For the induction step, let assume that the lemma is true up to $Y.rd = \rho \geq r$, and let us prove it for $\rho + 1$. To this end, we have to show that $Y.va\ell = X.va\ell$ for every Y that is written in REG with $Y.rd = \rho + 1$. Let us assume by contradiction that $Y.va\ell \neq X.va\ell$ and let p_i be the first process that writes $\langle \rho + 1, -, -, Y.va\ell\rangle$ into REG. This happens at line 04 or 05. In all cases, this implies that, at this moment, $view_j$ is $\mathcal{H}(\langle \rho, -, -, Y.va\ell\rangle)$. But, according to the induction assumption, this implies $Y.va\ell = X.va\ell$, a contradiction which completes the proof of the lemma. $\square_{Lemma\ 9}$

4.5 Proof of the algorithm: using the previous lemmas

Lemma 10. *No two processes decide different values.*

Proof Let r be the smallest round in which a process decides, p_i and $va\ell$ being the deciding process and the decided value, respectively. There is a time τ at which $view_i^{\tau}$ is $\mathcal{H}(\langle r, \texttt{up}, \texttt{false}, va\ell\rangle)$. Due to Lemma 9, every homogeneous snapshot starting from round r is necessarily associated with the value $va\ell$. Therefore, only this value can be decided in any round higher than r. Since r was assumed to be the smallest round in which a decision occurs, the consensus agreement property follows. $\square_{Lemma\ 10}$

Lemma 11. *For every quadruplet X that is written in REG, $X.va\ell$ is a value proposed by some process.*

Proof Let us assume by contradiction that $X.va\ell = v$ was not proposed by a process, and let p_i be the first process that writes X into REG. We consider two cases according to the line at which the write occurs.

- v is written into REG at line 04 or line 05. In this case, p_i obtained a view of REG in which at least some register contains the value v. According to the predicate of these two lines, the round number associated with v is necessarily greater than 0 which implies that v was previously written into REG and was not there initially. But this means that p_i is not the first process which writes v into REG, a contradiction.
- v is written into REG at line 08. In this case, the quadruplet X, where $X.val = v$, was returned by the call of the function $\mathsf{sup}()$, namely $\mathsf{sup}(view[1], \cdots, view[n], \langle 1, \mathtt{down}, \mathtt{false}, v_i\rangle)$, from which it follows that v is either v_i (the proposal of p_i) or some value that was previously written by another process. But, by assumption, p_i is assumed to be the first process to write v. Hence, $v = v_i$, which concludes the proof of the lemma. $\square_{Lemma\ 11}$

Lemma 12. *A decided value is a proposed value.*

Proof If a process decides a value v, it does it at line 03. Hence, according to the predicate of line 03, the round number associated with this value is greater than 0 which means that v was necessarily written into REG by some process. It then follows from Lemma 11, that v was proposed by a process, which establishes the claim. $\square_{Lemma\ 12}$

Lemma 13. *Let T be a non-empty set of quadruplets. For every $T' \subseteq T$: $\mathsf{sup}(T' \cup \{\mathsf{sup}(T)\}) = \mathsf{sup}(T)$.*

Proof Let $S = \mathsf{sup}(T)$. Hence $S.rd$ is the highest round number in T. Moreover, S is greater than, or equal to, any quadruplet in T. Hence, $\mathsf{max}(T' \cup \{S\}) = S$. Therefore, combined with the the definition of $\mathsf{sup}()$, we have: $\mathsf{sup}(T' \cup \{S\}) = \langle S.rd, S.\ell v\ell, conf\ell ict(T' \cup \{S\}), S.val\rangle$. Thus, in order to prove that $\mathsf{sup}(T' \cup \{S\}) = S$, we need to show that $conf\ell ict(T' \cup \{S\}) = S.cf\ell$.

There are two cases depending on the value of $S.cf\ell$.

- $S.cf\ell = \mathtt{true}$.
  As $S = \mathsf{max}(T' \cup \{S\})$ and due to the definition of $conf\ell ict()$ (line S3 in Figure 1), $S.cf\ell = \mathtt{true}$ implies that $conf\ell ict(T' \cup \{S\}) = \mathtt{true}$.
- $S.cf\ell = \mathtt{false}$.
  Since $S = \mathsf{sup}(T)$, $S.cf\ell = \mathtt{false}$ implies that $|tuples(T) = 1|$. Thus, S has a round number that is strictly greater than any other element of T. As $T' \subseteq T$, it follows that S is the only quadruplet in $T' \cup \{S\}$ with a round number equal to $S.rd$. Hence, $|tuples(T' \cup \{S\}) = 1|$. Since we assumed $S.cf\ell = \mathtt{false}$ and $S = \mathsf{max}(T' \cup \{S\})$, it follows that $conf\ell ict(T' \cup \{S\}) = \mathtt{false}$.

From the above case analysis, we conclude that $conf\ell ict(T' \cup \{S\}) = S.cf\ell$. $\square_{Lemma\ 13}$

Lemma 14. *If there is a time after which a process executes solo, it decides a value.*

Proof Assume that p_i eventually runs solo, we need to show that p_i decides. There exists a time τ, after which no other process than p_i writes into REG. Let $\tau' \geq \tau$ be the first time at which p_i takes a snapshot after τ. This snapshot is well defined, as p_i runs solo after τ and the implementation of atomic snapshot is obstruction-free. Let $S = \mathsf{sup}(view_i^{\tau'}[1], \cdots, view_i^{\tau'}[n], \langle 1, \mathtt{down}, \mathtt{false}, v_i\rangle)$.

Let us first show that there is a time after τ at which REG is $\mathcal{H}(S)$.

- If $REG^{\tau'}$ is $\mathcal{H}(S)$, we are done.
- If $REG^{\tau'}$ is not $\mathcal{H}(S)$, p_i executes line 06 and computes S. Then it writes S in an entry of REG (containing a value different from S), and re-enters the loop. If REG is then $\mathcal{H}(S)$, we are done. Otherwise, p_i executes again line 06 and, due to Lemma 13, the quadruplet computed by the function $\mathsf{sup}()$ is equal to S. It follows that after a finite number of iterations of the loop, REG is $\mathcal{H}(S)$.

When REG is $\mathcal{H}(S)$, we have the following.

- If $S = \langle -, \mathtt{up}, \mathtt{false}, - \rangle$, p_i decides in line 03.
- If $S = \langle r, \mathtt{down}, \mathtt{false}, val \rangle$, then p_i writes $Y = \langle r+1, \mathtt{up}, \mathtt{false}, val \rangle$ in line 04. Using the same argument as above, there is a time at which REG becomes $\mathcal{H}(Y)$, and the previous case holds.
- If $S = \langle r, -, \mathtt{true}, val \rangle$, then p_i writes $Y = \langle r+1, \mathtt{down}, \mathtt{false}, val \rangle$ in line 05. Then p_i keeps writing Y in the following iterations until REG becomes $\mathcal{H}(Y)$, and the previous case holds.

Hence, in all cases p_i eventually decides. $\square_{Lemma\ 14}$

Lemma 15. *If a single value is proposed, all correct processes decide.*

Proof Let us assume that all processes propose the same value v. It follows that all the processes keep writing $X = \langle 1, \mathtt{down}, \mathtt{false}, v \rangle$ until REG becomes $\mathcal{H}(X)$. Then, once every register of REG has been updated at least once, the processes start writing $Y = \langle 2, \mathtt{up}, \mathtt{false}, v \rangle$ until REG becomes $\mathcal{H}(Y)$ and v. When this occurs, v is decided. $\square_{Lemma\ 15}$

Theorem 1. *The algorithm of Figure* 2 *solves the obstruction-free consensus problem.*

Proof The proof follows directly from the Lemma 10 (Agreement), Lemma 12 (Validity), Lemma 14 (OF-Termination), and Lemma 15 (SV-Termination). $\square_{Theorem\ 1}$

5 From Consensus to (n, k)-Set Agreement

The algorithm The previous obstruction-free consensus algorithm of Figure 2 provides us "for free" with an obstruction-free (n, k)-set agreement algorithm. The only difference lies in the size of the snapshot object REG, which is now an array of $(n-k+1)$ MWMR atomic registers instead of an array of n MWMR atomic registers.

Theorem 2. *Assuming an underlying snapshot object composed of* $(n-k+1)$ MWMR *atomic registers, the algorithm of Figure* 2 *solves the obstruction-free* (n, k)*-set agreement problem.*

Proof The arguments for the validity and liveness properties are the same as the ones of the consensus algorithm since they do not depend on the size of REG.

As far as the k-set agreement property is concerned (no more than k different values are decided), we have to show that $(n - k + 1)$ registers are sufficient. To this end, let us consider the $(k - 1)$ first decided values, where the notion "first" is defined with respect to the linearization time of the snapshot invocation (line 02) that immediately precedes the invocation of the corresponding deciding statement (return() at line 03). Let τ be the time just after the linearization of these $(k - 1)$ "deciding" snapshots. Starting from τ, at most $(n - (k - 1)) = (n - k + 1)$ processes access the array REG, which is made up of exactly $(n - k + 1)$ registers. It follows that, after time τ, these $(n - k + 1)$ processes execute the algorithm of Figure 2, with the help of a snapshot object of size $(n - k + 1)$. Hence, from τ, these at most $(n-k+1)$ processes execute actually an anonymous obstruction-free consensus algorithm, during which they can decide at most one more value. It follows that at most k values are decided by the n processes. $\square_{Theorem\ 2}$

6 From One-shot to Repeated (n, k)-Set Agreement

6.1 The repeated (n, k)-set agreement problem

In the repeated (n, k)-set agreement problem, processes execute a sequence of (n, k)-set agreements. More precisely, a process invokes sequentially the operation propose$(1, v)$, then propose$(2, v')$, etc., where $1, 2, \ldots$ is the sequence number of the (n, k)-set agreement instance, and $v, v', \ldots$ is the value the process proposes to this instance.

It would be possible to associate a specific instance of the base algorithm described in Figure 2 with each sequence number, but this would require $(n - k + 1)$ atomic read/write registers per instance. The next section shows that in fact the repeated problem can be solved with only $(n - k + 1)$ atomic registers. According to the complexity results of [15], it follows that this algorithm is optimal in regard to the number of atomic read/write registers it uses. This closes the discussion regarding the space complexity of the repeated form of the (n, k)-set agreement problem.

6.2 Adapting the algorithm

From quadruplets to sextuplets Instead of a quadruplet, a register now contains a sextuplet $X = \langle sn, rd, \ell v\ell, cf\ell, va\ell, dcd\rangle$. The four fields $X.rd$, $X.\ell v\ell$, $X.cf\ell$, and $X.va\ell$ are the same as before. The additional field $X.sn$ is a sequence number. The other additional field $X.dcd$ is an initially empty list. From a rotational point of view, the jth element of this list is written $X.dcd[j]$; it contains a value decided at the jth instance of the repeated (n, k)-set agreement problem.

The total order over the sextuplets "$>$" is as previously lexicographical but now applies to the six fields. In particular, given two lists dcd and dcd', the relation $dcd > dcd'$ holds when there exists $j \geq 1$ such that $dcd[j] > dcd'[j]$ and for every $1 \leq k < j$, it is true that $dcd[k] \geq dcd[k']$. Relation "$\sqsupset$" is defined as follows:

$$X \sqsupset Y \stackrel{def}{=} (X > Y) \wedge [(X.sn > Y.sn) \vee (X.rd > Y.rd) \vee (X.cf\ell)].$$

Local variables Each process p_i now manages two local variables whose scope is the whole repeated (n, k)-set agreement problem. (Hence, we do no longer have the locally memoryless property of the base obstruction-free algorithm presented in Figure 2.)

- The variable sn_i, initialized to 0, is used by p_i to generate its sequence numbers. It is assumed that p_i increments sn_i before invoking propose(sn_i, v_i).
- The local list dcd_i is used by p_i to store the value it has decided during the previous instances of the (n, k)-set agreement. Hence, $dcd_i[sn]$ contains the value decided by p_i during the snth instance. These lists are exchanged by the processes, which allows the slower of them to catch up when they are in late.

The algorithm Figure 3 describes the algorithm executed at some process p_i. The new parts, with respect to the base algorithm in Figure 2, are underlined and in blue. To ease the understanding, both algorithms use the same line numbering. Figure 3 contains a single new line, marked with "N". Suppressing all the underlined parts of the new algorithm leads to our base solution. In what follows, we detail the internals of our solution then establish its correctness.

- Line 03. When all the entries of the view obtained by p_i contain only sextuplets whose first five fields are equal, p_i decide the value $va\ell$. But before returning $va\ell$, p_i writes $va\ell$ in $dcd_i[sn_i]$. The idea is that, when p_i will execute the next (n, k)-set agreement instance (whose sequence number will be $sn_i + 1$), it will be able to help processes whose current sequence number is smaller than sn_i.

```
operation propose(sn_i, v_i) is
(01) repeat forever
(02)   view ← REG.snapshot();
(03)   case (∃r > 0, val : ∀z : view[z] = ⟨sn_i, r, up, false, val, −⟩) then
                  dcd_i[sn_i] ← val;
                  return(val);
(04)       (∃r > 0, val : ∀z : view[z] = ⟨sn_i, r, down, false, val, −⟩) then
                  REG.write(1, ⟨sn_i, r + 1, up, false, val, dcd_i⟩);
(05)       (∃r > 0, val, level : ∀z : view[z] = ⟨sn_i, r, level, true, val, −⟩) then
                  REG.write(1, ⟨sn_i, r + 1, down, false, val, dcd_i⟩);
(06)       otherwise let ⟨inst, r, level, conflict, val, dec⟩
                              ← sup(view[1], · · · , view[n], ⟨sn_i, 1, down, false, v_i, dcd_i⟩);
(N)              if (inst > sn_i) then dcd_i[sn_i] ← dec[sn_i]; return dcd_i[sn_i] end if;
(07)             z ← smallest index y such that view[y] = min(view[1], · · · , view[n]);
(08)             REG.write(z, ⟨inst, r, level, conflict, val, dec⟩);
(09)   end case
(10) end repeat.
```

Figure 3: Repeated anonymous obstruction-free consensus

- Line 04. Process p_i obtains a quadruplet of the form $\langle sn_i, r, \mathtt{down}, \mathtt{false}, val, -\rangle$. In such a case, p_i writes $\langle sn_i, r, \mathtt{down}, \mathtt{false}, val, dcd_i\rangle$ to $REG[1]$. (Let us notice here that the write of dcd_i is to help other processes deciding (n, k)-set agreement instances whose sequence number is smaller than sn_i.)
- Line 05. Similar to the previous case, except that a conflict now appears in the view computed by the process p_i.
- Lines 06-10. Process p_i computes the supremum of the snapshot $view$ obtained at line 03, as well as the sextuplet $\langle sn_i, 1, \mathtt{down}, \mathtt{false}, val, dcd_i\rangle$. There are two cases to consider.
  - If the sequence number of the supremum is greater than sn_i, process p_i benefit from the list of decisions stored in the supremum. More precisely, this help is obtained from the item $dec[sn_i]$. Similarly to line 03, process p_i then writes this decision in $dcd_i[sn_i]$ before returning from its call.
  - In the other case, the sequence number of the supremum is equal to sn_i. Process p_i then executes the same operations as in the basic algorithm (lines 07-08).

6.3 Proof of the algorithm

This section first presents a simple technical lemma, and then shows that the algorithm described in Figure 3 solves the repeated (n, k)-set agreement problem.

Lemma 16. *For any $m \geq 1$, if X is written in REG and $X.sn = m$, then for every $1 \leq k < m$, $X.dcd[k]$ exists.*

Proof Let X be some sextuplet in REG for which $X.sn = m$ holds. Name p_i the process that first writes X. The operation of p_i might occur either at line 04, 05 or 08. By definition of the repeated (n, k)-set agreement, if p_i starts instance m of the problem, than it has already returned from the prior instances. As a consequence, if the write occurs at line 04 or 05, then the invariant holds. Now in the case where the write takes place at line 08, the definition of function sup() tells us that either some process wrote a quadruplet Y with $Y.sn = m$ previously, or $X = \langle sn_i, 1, \mathtt{down}, \mathtt{false}, v_i, dcd_i\rangle$. In the former case, we repeat our previous reasoning. In the later, we know that the invariant holds since p_i has already taken a decision in the instances prior to m. □*Lemma 16*

Theorem 3. *The algorithm in Figure 3 solves the obstruction-free repeated (n, k)-set agreement problem.*

Proof Let us consider some execution ρ of the algorithm in Figure 3. Let m be an instance that was executed in ρ. We name U_m the set of decisions taken at instance m in ρ. We also define $V_m \subseteq U_m$ the decisions taken at instance m in ρ before a higher instance begins, that is before some sextuplet X with $X.sn > m$ is observed by a process in REG.

First of all, let us notice that, if no interleaving takes place between m and a higher instance, the algorithm of Figure 3 boils down to our base algorithm. As a consequence, decisions in V_m are valid and $|V_m| \leq k$.

Let us now choose some decision $u \in U_m$ taken by a process p_i in ρ. Below, we show that $u \in V_m$ holds.

- If the decision takes place at line 03, then it occurs before an instance higher than m begins. Thus, by definition $u \in V_m$ holds.
- Let us now consider the case where process p_i decides at line N with $inst > m$, choosing $dec[m]$ as its decision. Lemma 16 tell us that this value is well-defined. In Figure 3, we observe that a process p_j might update $dcd_j[m]$ with value u only in the case where it decides u in instance m at line 03. Thus, u belongs to V_m.

The previous reasoning shows that every instance is safe, in the sense that it satisfies both the validity and agreement properties of the (n, k)-set agreement problem.

In Figure 3, we observe that either a process decides before a higher instance begins, or it decides immediately afterward. As a consequence, the properties of OF-termination and SV-termination follow from the validity property and the fact that our base solution satisfies these two properties. $\square_{Theorem\ 16}$

Theorem 3 implies that solving repeated (n, k)-set agreement in an anonymous system does not require more atomic read/write registers than the base non-repeated version. The additional cost lies only in the size of the atomic registers which contain two supplementary unbounded fields. As pointed out at the beginning of this section, the lower bound established in [15] induces that the algorithm in Figure 3 is space-optimal.

7 From Obstruction-Freedom to x-Obstruction-Freedom

This section extends the base algorithm described in Figure 2 to obtain a solution to the anonymous (n, k)-set agreement problem with a stronger progress condition, namely x-obstruction-freedom. It first defines x-obstruction-freedom, then details the modifications to the base algorithm, and finally proves its correctness.

7.1 Problem statement

The notion of x-obstruction-freedom [38] guarantees that for every set of processes P, with $|P| \leq x$, every correct process in P returns from its operation invocation if no process outside of P takes a step for a "long enough" period of time. This progress property naturally generalizes obstruction-freedom, which corresponds to the case where $x = 1$. Moreover, x-obstruction-freedom and wait-freedom coincide in every n-process system where $x \geq n$. When $x < n$, x-obstruction-freedom depends on the concurrency pattern, while wait-freedom does not.

The variant of the (n, k)-set agreement problem in which we are interested is defined as follows. Its Validity, Agreement and SV-Termination properties remain the ones we stated in Section 2.2. Differently, OF-Termination is modified as follows:

- x-OF-termination. If there is a time after which at most x correct processes take steps, each of these processes eventually decides a value.

Let observe that every x-obstruction-free solution to (n, k)-set agreement is also a wait-free solution to $(k+1, k)$-set agreement when $x > k$. It then follows from [7, 24, 35] that there is no x-obstruction-free algorithm for $x > k$. As a consequence, the rest of this section assumes $x \leq k$.

7.2 Algorithm

The shared memory Under x-obstruction-freedom, up to x processes may concurrently progress without preventing termination. As a consequence, in comparison to obstruction-freedom, solving k-set agreement in this setting requires to deal with more contention scenarios. To cope with these additional interleavings of processes, we increase the number of entries in REG. More precisely, REG now contains $m = (n - k + x)$ entries.

Ordering the quadruplets In the base algorithm, the four fields of some quadruplet X are the round number $X.rd$, the level $X.\ell v\ell$, the conflict flag $X.cf\ell$, and the value $X.val$. Coping with x-concurrency requires to replace the last field, which was initially a singleton, with a set of values. Hereafter, this new field is denoted $X.valset$.

In line with the definitions of Section 4.1, let "$>$" denote the lexicographical order over the set of quadruplets, where the relation $\sqsupset$ is generalized as follows to take into account the fact that the last field of a quadruplet is now a non-empty set of values:

$$X \sqsupset Y \stackrel{def}{=} (X > Y) \wedge [(X.rd > Y.rd) \vee (X.cf\ell) \vee (X.valset \supseteq Y.valset)].$$

In comparison to the definition appearing in Section 4, the sole new case where the ordering $X \sqsupset Y$ holds is $(X > Y) \wedge (X.valset \supseteq Y.valset)$. This case captures the fact that, as long as at most x input values are competing at some round, there is no conflict. If such a situation arises, we simply construct a quadruplet that aggregates the different input values.

```
function sup(T) is                                  % T is a set of quadruplets whose last field is now a set of values %
(S1)  let ⟨r, level, conflict, valset⟩ be max(T);                          % lexicographical order %
(S2)  let tuples(T) be {X | X ∈ T ∧ X.rnd = r};
(S3)  let values(T) be {v | X ∈ T ∧ v ∈ X.valset};
(S4)  let conflict(T) be conflict ∨ |tuples(T)| > x ∨ |values(T)| > x;
(S5)  let valset be the (at most) x greatest values in values(T);
(S6)  return(⟨r, level, conflict(T), valset⟩).
```

Figure 4: Function sup() suited to x-obstruction-freedom

Modifications to the sup() function Figure 4 describes the new definition of function sup(). Compared with the original algorithm in Figure 1, it introduces a few modifications (underlined and in blue). Those are detailed below.

- Line S1. As pointed out previously, the last field of a quadruplet is now a set of values. The lexicographical ordering over such sets is as follows: sets are ordered first according to their size, and second using some arbitrary order over their elements. By abuse of notation, this order is also written $<$. For instance, we have $\{10, 8, 2\} < \{10, 4, 3\}$ and $\{10, 4, 3\} < \{15, 12\}$. It is assumed that for any set of values S, $S < \perp$ holds.
- Line S2. This line does not change.
- Lines S3 and S4. This variant extends the definition of a conflict. Namely, it considers as a conflict the case where more than x distinct tuples are competing at round r, and also the additional case where more than x distinct values are competing at round r.

- Lines S5 and S6. Compared with Figure 1, function sup() returns a quadruplet that may contain additional input values. This comes from the fact that the function aggregates the x greatest values competing at round r.

```
operation propose(v_i) is
(01) repeat forever
(02)     view ← REG.snapshot();
(03)     case (∃r > 0, valset : ∀z : view[z] = ⟨r, up, false, valset⟩) then
                 let val be any value in valset;
                 return(val);
(04)         (∃r > 0, valset : ∀z : view[z] = ⟨r, down, false, valset⟩) then
                 REG.write(1, ⟨r + 1, up, false, valset⟩);
(05)         (∃r > 0, valset, level : ∀z : view[z] = ⟨r, level, true, valset⟩) then
                 REG.write(1, ⟨r + 1, down, false, valset⟩);
(06)         otherwise let ⟨r, level, cfl, valset⟩
                                   ← sup(view[1], ··· , view[n], ⟨1, down, false, {v_i}⟩);
(07)                 z ← smallest index y such that view[y] ≠ ⟨r, level, cfl, valset⟩;
(08)                 REG.write(z, ⟨r, level, cfl, valset⟩);
(09)     end case
(10) end repeat.
```

Figure 5: Anonymous x-obstruction-free (n, k)-set agreement

Solving (n, k)-set agreement under x-obstruction-freedom Figure 5 presents the modified algorithm solving the (n, k)-set agreement problem, in which the progress condition is x-obstruction-freedom. Let us notice that it is also locally memoryless.

In Figure 5, the differences between the two algorithms are underlined and in blue. These differences come from the fact that, as detailed previously, each quadruplet now contains a set of input values in its last field. The main difference is at line 03.

- Line 03. When deciding, a process must pick one of the values provided in the snapshot taken from REG.

7.3 Correctness proof

This section proves that the algorithm described in Figure 5 is a correct solution to the (n, k)-set agreement problem, when considering x-obstruction-freedom as the progress condition. The proof scheme is similar to the one used in Section 4.

Theorem 4. *The algorithm in Figure 5 solves the x-obstruction-free (n, k)-set agreement problem.*

Proof Validity and SV-Termination follow from a reasoning identical to the one conducted for the base algorithm.

As far as the Agreement property is concerned, let us first observe that the relation $\sqsupset$ remains a partial order. Then, considering some non-empty set of quadruplets T and some $X \in T$, the rewriting of function sup() maintains that $\mathsf{sup}(T) \sqsupseteq X$. Indeed, we have $\mathsf{sup}(T) \geq \mathsf{max}(T)$ and either (*i*) $X.rd < \mathsf{sup}(T).rd$, or (*ii*) there is a conflict leading to $\mathsf{sup}(T).cfl = \mathtt{true}$, or (*iii*) since $|values(T)| \leq x$, we have $X.valset \subseteq \mathsf{sup}(T).valset$.

Then, let us consider a run where at most $(n - k + x)$ processes take steps. From what precedes, we may conclude that (excluding Lemmas 7 and 9) Lemmas 3 to 8 hold for the algorithm in Figure 5. The reformulations of Lemmas 7 and 9 as well as their respective correctness proofs are stated below.

Lemma 17. *If REG^{τ} is $\mathcal{H}(X)$, $REG^{\tau'}$ is $\mathcal{H}(Y)$, $\tau' \geq \tau$, $(Y.rd = X.rd)$ and $(\neg Y.cfl)$ then $(Y.valset \supseteq X.valset)$.*

Proof According to Lemma 6, we have $Y \sqsupseteq X$. If $Y = X$, then the claim follows immediately. On the other hand, if $Y \sqsupset X$ holds, since we know that $(Y.rd = X.rd)$ and $(\neg Y.cf\ell)$, the definition of $\sqsupseteq$ implies that $Y.valset \supseteq X.valset$. $\square_{Lemma\ 17}$

Lemma 18. $[(REG^\tau \textit{ is } \mathcal{H}(X)) \wedge (X.\ell v\ell = \texttt{up}) \wedge (\neg X.cf\ell) \wedge (REG^{\tau'} \textit{ is } \mathcal{H}(Y)) \wedge (Y.rd \geq X.rd)]$ $\Rightarrow (Y.valset \supseteq X.valset \vee X.valset \supseteq Y.valset)$.

Proof The proof is by induction on $Y.rd$. Let us first assume $Y.rd = X.rd$. There are two cases.

- (Case $\tau \geq \tau'$) As $X.cf\ell$ equals `false`, Lemma 17 implies that $X.valset \supseteq Y.valset$.
- (Case $\tau' > \tau$). From Lemma 6, we know that $Y \sqsupseteq X$. As $Y.rd = X.rd$, it follows that $Y.\ell v\ell \geq X.\ell v\ell = \texttt{up}$. Applying Lemma 8, we obtain the existence of two quadruplets $Z_X = \langle X.rd-1, \texttt{down}, \texttt{false}, X.valset\rangle$ and $Z_Y = \langle X.rd-1, \texttt{down}, \texttt{false}, Y.valset\rangle$ such that REG is $\mathcal{H}(Z_X)$ and $\mathcal{H}(Z_Y)$ at some points in time. From Lemma 17, we deduce that $X.valset \supseteq Y.valset$, or the converse, holds.

For the cases where $Y.rd > X.rd$, let us consider that our induction assumption holds up to some round r. Then, let p_i be the first process that writes at round $r+1$ some quadruplet $\langle r+1, -, -, Y.valset\rangle$ into REG. This happens at either line 04, or line 05 in Figure 5. In both cases, our induction assumption implies that $X.valset \supseteq Y.valset$ or the converse holds. $\square_{Lemma\ 18}$

Lemma 19. *At most x values are decided.*

Proof Let V be the set of decided values. Since each decision takes place at line 03, there exists a set $(X_v)_{v\in V}$ of tuples such that some process observes an homogeneous snapshot $\mathcal{H}(X_v)$ with $X_v.cf\ell =$ `false`, $X_v.\ell v\ell = \texttt{up}$ and $v \in X_v.valset$.

For any tuple X_v, since $X_v.cf\ell = \texttt{false}$, $|X_v.valset| \leq x$ holds. From Lemma 18, we deduce that for any two values $v, w \in V$, either $X_v.valset \subseteq X_w.valset$, or the converse, holds. The conjunction of the above two observations implies that $|V| \leq x$. $\square_{Lemma\ 19}$

Applying the reasoning of Section 5, Lemma 19 implies that the algorithm described in Figure 5 satisfies the Agreement property of (n, k)-set agreement: $n - (n - k + x)$ processes may decide up to $(k - x)$ values, and the $(n - k + x)$ remaining processes decide at most x values.

The lemmas that follow establish the x-OF-termination property. To this end, and in line with the definition of x-OF-termination, we consider some set of processes P_x, with $|P_X| \leq x$, and a run λ of the algorithm in Figure 5 satisfying $\lambda = \lambda_1\lambda_2$, and only the processes of P_x take steps during λ_2. If $x = 1$, then the algorithm in Figure 5 boils down to our base solution. As a consequence, we assume hereafter $x \geq 2$.

Lemma 20. *All the correct processes decide in λ, or for every round r, every entry of REG contains eventually some tuple X with $X.rd > r$.*

Proof We proceed by contradiction. Let $P = \{p_1, \ldots, p_m\}$ be the largest set of processes that never decide. Consider a point in time τ_0 where only the processes in P take steps. At time $\tau + 1 > \tau_0$ and for every process $p_i \in P$,

- if p_i modifies $view_i$, then this corresponds to the value of REG^τ (line 02); and
- if p_i writes some tuple X in REG, then $X \geq \mathsf{sup}(view_i^\tau)$ (lines 04, 05 and 08).

Let S^τ be the set $\{\mathsf{sup}(view_1^\tau), \ldots, \mathsf{sup}(view_m^\tau), \mathsf{sup}(REG^\tau)\}$, and define m^τ the tuple $min(S^\tau)$. The above observation implies that $(m^\tau)_{\tau\geq\tau_0}$ is growing.

By assumption, there exists some round r and an entry $REG[i]$ such that $REG[i]$ never contains a tuple X with $X.rd > r$. At the light of the code in Figure 5, every entry $REG[j]$ should satisfy

$REG[j].rd \leq r$ at all time. From what precedes, the sequence $(m^\tau)_{\tau \geq \tau_0}$ is upper bounded and converges toward some value m.

Note τ_1 the time at which the convergence takes places, that is $m^\tau = m$ is true for every $\tau \geq \tau_1$. After time τ_1, every new write $\mathcal{W}(-, X)$ to REG is such that $X \geq m$. As a consequence, m can be written to an entry of REG at most $|P|$ times after time τ_1 (see lines 07 and 08 in Figure 5). It follows that eventually it is true forever that $\mathsf{sup}(REG^\tau) > m$ holds, or that all the entries of REG contains m. Both cases leads to a contradiction: The former case is not possible, since m is the limit of $(m^\tau)_{\tau \geq \tau_0}$. In the later, a process eventually observes an homogeneous snapshot for m. Since this process cannot decide, it executes line 04 or 05, writing some tuple X in REG with $X.rd = m.rd + 1$. $\square_{Lemma\ 20}$

Lemma 21. *All the processes of P_x decides in λ.*

Proof Execution λ is such that $\lambda = \lambda_1\lambda_2$ and only the processes of P_x take steps in λ_2. Lemma 20 tells us that either all the processes in P_x decide, or λ_2 contains an unbounded amount of rounds, starting from some round r_0. Let us consider the later case, assuming without lack of generality, that none of the processes in P_x decide in λ_2.

For some round r, we note $\mathcal{X}_r$ the set $\{X : \exists p_i, \tau : view_i^\tau \text{ is } \mathcal{H}(X) \wedge X.rd = r\}$. We also define V_r as $min(\{V : \exists X \in \mathcal{X}_r \ \wedge \ X.valset = V\})$. In what follows, we state several claims regarding the sequence $(V_r)_{r > r_0}$.

- Claim C1. $\forall X, r : (\exists \tau, j : REG^\tau[j] = X \wedge X.rd = r + 1) \Rightarrow X.valset \geq V_r$.

  Proof. The above equation is true initially, as for any i, $REG^0[i] = \langle 0, \mathtt{down}, \mathtt{false}, \perp \rangle$. Then, consider that it holds up to time τ, and assume that a process p_i writes a tuples X in REG with $X.rd = r + 1$ at time $\tau + 1$. If p_i executes line 04 or 05, then there exists $Y \in \mathcal{X}_r$ with $X.valset = Y.valset \geq V_r$. Otherwise p_i executes line 08 and in such a case $X = \mathsf{sup}(view_i^\tau)$. Observe that for any $Z \in view_i^\tau$ satisfying $Z.rd = r + 1$, line S5 in function $\mathsf{sup}()$ implies that $X.valset \geq Z.valset$, and from our induction assumption, $Z.valset \geq V_r$. Thus, $X.valset \geq V_r$ holds. $\square$

- Claim C2. $\forall X, r : (\exists \tau, j : REG^\tau[j] = X \wedge X.rd = r + 1 \wedge X.valset = V_r) \Rightarrow X.cf\ell = \mathtt{false}$.

  Proof. The above equation is initially true. In what follows, we consider that it holds up to time τ, and assume that at time $\tau + 1$ a process p_i writes some tuple X in REG with $X.rd = r + 1 \ \wedge \ X.valset = V_r$.

  For the sake of contradiction, consider that $X.cf\ell = \mathtt{true}$. In Figure 5, a write to REG might occur either at line 04, 05 or 08. Since $X.cf\ell = \mathtt{true}$ holds, p_i executes line 08 at time $\tau + 1$ with $X = \mathsf{sup}(view_i^\tau)$. Define $M = max(T)$, with $T = \{Z \in view_i^\tau : Z.rd = r + 1\}$. From the code at lines S5 and S6 in Figure 4, $M.rd = X.rd = r + 1$ and $X.valset \geq M.valset$ are both true. Claim C1 implies that $M.valset \geq V_r$. As $X.valset \geq M.valset$ and $X.valset = V_r$, necessarily $M.valset = V_r$. Then, applying our induction assumption, we deduce that $M.cf\ell = \mathtt{false}$. Since $X.cf\ell = \mathtt{true}$, either $|tuple(T)| > x$ or $|values(T)| > x$ holds. Below, we explore each of these two cases.

  - $(|values(T)| > x)$ For every $Z \in T$, we have $|Z.valset| \leq x$. Hence, there exist $Y, Z \in T$ such that $Y.valset \neq Z.valset$. By C1, both $Z.valset \geq V_r$ and $Y.valset \geq V_r$ are true. It follows, that $X.valset \geq M.valset \geq max(Y.valset, Z.valset) > V_r$. Contradiction.
  - $(|tuple(T)| > x)$ Consider some $Z \in T$. If $Z.valset = V_r$, our induction assumption implies that $Z.cf\ell = \mathtt{false}$ holds. Thus, there are at most two tuples in T of the form $\langle r + 1, -, \mathtt{false}, V_r \rangle$. As $x \geq 2$, T contains at least three elements. It follow from C1 that there exists some $Z \in T$ with $Z.valset > V_r$. From which, we deduce that $M.valset > V_r$. Contradiction.

□

- Claim C3. $\forall r > r_0 : V_r < V_{r+2}$.

  Proof. Claim C1 implies that $V_r \leq V_{r+1} \leq V_{r+2}$. Consider that for some round r, the assertion $V_r = V_{r+1} = V_{r+2}$ holds. Choose X and Y respectively in $\mathcal{X}_{r+1}$ and $\mathcal{X}_{r+2}$, with $X.valset = Y.valset = V_r$. From claim C2, $X.cf\ell =$ `false` holds. Applying C1, a short induction tells us that every tuple Z with $(Z.valset = V_r \wedge Z.rd = r + 2)$ satisfies $Z.\ell v\ell =$ `up`. This implies that $Y.\ell v\ell =$ `up` is true. Since C2 holds also for the tuple Y, $Y.cf\ell =$ `false` holds. As a consequence, some process decides at round $r + 2$. This contradicts that $r + 2 > r_0$ is true. □

If the number of rounds is not bounded, then the claim C3 implies that $(V_r)_{r>r_0}$ diverges. However, any element in this sequence is a subset of the values proposed at round r_0, contradicting such an assumption. $\square_{Lemma\ 21}$

It follows from Lemmas 19 and 21 that the algorithm presented in Figure 5 solves the x-obstruction (n, k)-set agreement problem. $\square_{Theorem\ 4}$

8 On The Power of Repeated Anonymous Consensus

8.1 Universality of n MWMR registers

Let us first turn our attention to non-anonymous systems made up of n asynchronous processes communicating with SWMR read/write registers. Let us first notice that, if an object can be implemented with an arbitrary number of SWMR atomic read/write registers, it can be implemented with only n SWMR atomic read/write registers, one per process. This follows from the observation that, for every writer p_i, we can glue together all the SWMR atomic read/write registers that p_i accesses into an array stored in a single register. At the light of this result, we raise the question whether a similar result exists in the context of anonymous systems. This section answers positively this question, showing that what can be obstruction-free computed by n anonymous processes with an arbitrary number of MWMR atomic registers, can also be obstruction-free computed by n anonymous processes with no more than n MWMR atomic registers.

Theorem 5. *Let O be an object that can be obstruction-free implemented by n anonymous processes and any number of* MWMR *atomic read/write registers. O can be obstruction-free implemented by n anonymous processes and n* MWMR *atomic read/write registers.*

Proof The proof consists in building a simple universal construction whose core is the obstruction-free anonymous repeated consensus algorithm presented in Section 6. Let (a) p_1, ..., p_n be the application processes, (b) $R1$, $R2$, etc., be the MWMR atomic read/write registers they share (there registers implement object O), and (c) in_1, ..., in_n be the individual inputs of the n processes. Let q_1, ..., q_n be a set of n anonymous simulators.

Each simulator q_i is assigned exactly one process p_i. Moreover, the local memory of each simulator contains a copy of all the registers $R1$, $R2$, etc. The memory shared by the simulators contains only the snapshot object on which is built the obstruction-free anonymous repeated consensus algorithm. Hence, it is made up of n atomic read/write registers.

Each application process executes a sequence of steps, that is a sequence of read and write operations on the registers $R1$, $R2$, etc.

The simulation is a sequence of repeated consensus, which proceeds as follows.

- First consensus. Each simulator q_i proposes the value $prop_i$ ="step of process with input in_i is: read register Rx" (or "'step of process with input in_i is: write v in register Ry"), where "read register Rx" or " write v in register Ry" is the first step of p_i. Let dec be the value decided by this first consensus instance. There are two cases.
  - If $dec = prop_i$, q_i applies locally the operation "read register Rx" (or "write v in register Ry"), and prepares (for the next consensus instance) a new proposal $prop_i$ according to next step of p_i as defined by its code, namely, $prop_i$ is "step of process with input in_i is: ...".
  - If $dec \neq prop_i$, q_i keeps its proposal $prop_i$ for the next consensus instance, but modifies its local copy of Ry if dec is "write v in register Ry".
- Second consensus. Each simulator q_i proposes the value $prop_i$ it computed after it terminated the first consensus instance, and proposes it to the second consensus instance.
- And so on.

Let us observe that, as several processes can have the same input and anonymous processes have the same code, it is possible that several simulators propose the same value $prop$ to a consensus instance, where $prop$ is "step of process with input in_i is read register Rx" (or "step of process with input in_i is write v register Ry". If such a $prop$ is decided by the corresponding consensus instance, and the decided value is "write v in register Ry", only one write is applied to Ry by each simulator. This does not create a problem, as this write of v in R_y can be seen as a digest of the corresponding number of consecutive writes of v in R_y, each one overwriting the previous one.

It follows from the sequence of repeated consensus that all simulators see the same sequence of steps issued by the processes, which is a linearization of the operations issued by the processes. Moreover, each process p_i inherits the progress of its simulator q_i. Hence, if a simulator q_i executes alone a long enough period of time to compute an output, so does the corresponding simulated process p_i, which concludes the proof of the theorem. $\square_{Theorem\ 5}$

8.2 Anonymity, tasks, and colorless tasks

The previous theorem showed that, in an anonymous system, n registers are sufficient to obstruction-free implement any object O implemented with more registers. An interesting follow-up question is to know which distributed tasks (see below), usually considered in a non-anonymous system, can be solved in an anonymous setting. As we shall see below, this set contains at least all colorless tasks, i.e., the distributed tasks that do not involve some kind of symmetry breaking argument.

Distributed tasks A distributed task T is defined by a set $\mathcal{I}$ of input n-vectors, a set $\mathcal{O}$ of output n-vectors and a map Δ from $\mathcal{I}$ to $2^{\mathcal{O}}$. If the input value of a process p in $I \in \mathcal{I}$ is $\perp$, we say that p does not participate to the input vector I. Similarly if $O[p]$ equals $\perp$, process p does not decide in O. For every distributed task $T = (\Delta, \mathcal{I}, \mathcal{O})$, we require that *(i)* a process may not decide $((\forall p : O'[p] \in \{O[p], \perp\} \wedge (I, O) \in \Delta) \Rightarrow (I, O') \in \Delta)$, and that *(ii)* a process that does not participate, does not decide $((I[p] = \perp \wedge (I, O) \in \Delta) \Rightarrow O[p] = \perp)$.

The notion of *interval linearizability* (which generalizes linearizability [26]) was introduced in [10], where it is shown that tasks and one-shot concurrent objects are in a precise sense equivalent (Theorem 3 in [10]). It follows that the proof of the previous theorem remains correct if we consider a distributed task instead of an object O. We have consequently the following theorem.

Theorem 6. *if a distributed task T can be obstruction-free solved by n anonymous processes and any number of MWMR atomic read/write registers, T can be obstruction-free solved by n anonymous processes with no more than n MWMR atomic read/write registers.*

The case of colorless tasks Let us note $val(U)$ the set of non-null values in some vector U. Following [8], a task $T = (\Delta, \mathcal{I}, \mathcal{O})$ is *colorless* when in a solution to T, a process is free to adopt the input and

output value of any other participating process. Formally, $((val(I) \subseteq val(I') \wedge val(O') \subseteq val(O) \wedge (\forall p : I'[p] = \perp \Rightarrow O'[p] = \perp) \wedge (I, O) \in \Delta) \Rightarrow (I', O') \in \Delta)$. This class of distributed tasks includes notably the k-set agreement problem. The next theorem shows that anonymity is enough when a task is colorless and obstruction-free -solvable.

Theorem 7. *Let us consider a colorless task $T = (\Delta, \mathcal{I}, \mathcal{O})$ that is obstruction-free solvable in a non-anonymous system using any number of* SWMR *registers. Then, T is also obstruction-free solvable in an anonymous system with n* MWMR *atomic registers.*

Proof Let us consider an obstruction-free algorithm A, which solves T in a non-anonymous system. As n registers are sufficient in such a setting, we assume (without lack of generality) that A uses n registers only. In what follows, we present a construction to simulate a run of A in an anonymous system, and then proves its correctness.

> Construction. First of all, each anonymous process p proposes $(0, v)$ to consensus, where v is its input value. Upon deciding some tuple (i, w), if $w = v$, then process p considers that it holds identifier i; otherwise p computes $i' = i + 1$ and proposes (i', v) to the next consensus instance. This process repeats until p holds some identifier. Then, process p executes algorithm A with input v and identifier i. As in the proof of Theorem 5, all the processes holding identifier i agree on simulating the next step of process i with the help of anonymous consensus. During this simulation, we note that registers are SWMR and in particular that process i writes only to register $R[i]$. Process p decides the value the simulation of process i outputs.

Let us now show that this construction is correct. To this end, consider some input vector $I \in \mathcal{I}$ and a run ρ following the above algorithm.

- If no decision occurs in ρ, the output vector O that contains $\perp$ everywhere satisfies $(I, O) \in \Delta$.
- Assume now that a process p proposes a value u and decides some value v. Before process p decides, it must have chosen some identifier i. At the light of the above construction, all the processes that have identifier i propose and decide the same value. Hence, in run ρ, the simulated process i proposes u and decides v. Generalizing this reasoning, let us note I' and O' respectively, the simulated input and output n-vectors during the simulation.
  Since T is colorless, obstruction-free solvable, and in an asynchronous system we cannot distinguish a non-participating process from an initially crash one, vector I' belongs to the domain of Δ. Then, we observe that in ρ the identifiers of any two simulated processes are different. This ensures that the simulated system is non-anonymous. Consequently, A solves T in ρ and $O' \in \Delta(I')$ holds.
  Let O be the n-vector output in ρ. By construction, $val(O) \subseteq val(O')$ holds. As T is colorless, we deduce that O belongs to $\Delta(I')$. Then, since $val(I') \subseteq val(I')$, T is colorless and $O \in \Delta(I')$, we conclude that $O \in \Delta(I)$.

To complete the proof, let us observe that all the steps in the above construction are obstruction-free, and that (as pointed out) the n-vector input in the simulation belongs to the domain of Δ. As a consequence, if T is obstruction-free solvable in a non-anonymous system, it remains obstruction-free solvable in an anonymous system. $\square_{Theorem\ 7}$

9 Conclusion

This paper first presented a one-shot obstruction-free (n, k)-set agreement algorithm for a system made up of n asynchronous anonymous processes that communicate with atomic read/write registers. This algorithm uses only $(n - k + 1)$ registers. In terms of the number of registers, it is the best algorithm known so far, and, in the case of consensus, it is up to an additive factor of 1 close to the best known

lower bound [41]. This algorithm answers the challenge posed in [13], and establishes a novel upper bound of $(n-k+1)$ on the number of registers to solve one-shot obstruction-free (n,k)-set agreement. This upper bound improves the ones stated in [15] for anonymous and non-anonymous systems.

Further, the paper introduced a simple extension of the basic algorithm, that solves repeated (n,k)-set agreement. The lower bound of $(n-k+1)$ atomic registers was established in [15] for this problem. Hence, the proposed algorithm proves that this bound is tight. A one-shot algorithm solving anonymous (n,k)-set agreement problem in the context of x-obstruction-freedom has also been described. This algorithm makes use of $(n-k+x)$ atomic read/write registers.

All these algorithms rely on the same round-based data structure. The basic one-shot algorithm does not require persistent local variables, and in addition to a proposed value, an atomic register solely contains two bits and a round number. The algorithm solving repeated (n,k)-set agreement requires that each atomic register includes two additional fields.

The paper has also presented two reduction results. The first one showed that any distributed task that is obstruction-free solvable in an anonymous system with an arbitrary number of registers is also obstruction-free solvable with solely n registers. The second reduction showed that this amount of registers is also enough for every colorless task which is obstruction-free solvable in a non-anonymous system.

Let the MWMR-*number* of a concurrent object O be the minimal number of MWMR atomic registers needed to implement O in an n-process asynchronous anonymous system in which any number of processes may crash. Using this terminology, it is shown in [15] that the MWMR-number of the repeated obstruction-free (n,k)-set agreement object is at least $(n-k+1)$. Showing that this number is actually $(n-k+1)$, this paper closes the corresponding lower bound problem. Furthermore, Theorem 5 shows that no object (defined by a sequential specification) has an MWMR-number greater than n. Finally, we conjecture that $(n-k+1)$ is the MWMR-number of the one-shot obstruction-free (n,k)-set agreement object, and more generally that $(n-k+x)$ is the MWMR-number of one-shot x-obstruction-free (n,k)-set agreement objects, when $1 \leq x \leq k < n$.

Acknowledgments

The authors want to thank Rati Gelashvili for fruitful comments on a preliminary version of the paper, which gave rise to Section 8. They want to thank also the referees for their constructive comments, which helped them improve and simplify the presentation.

This work was partially supported by the Franco-German DFG/ANR project DISCMAT devoted to connections between mathematics and distributed computing, and the French ANR project DESCARTES devoted to distributed software engineering.

References

[1] Afek Y., Attiya H., Dolev D., Gafni E., Merritt M., and Shavit N., Atomic snapshots of shared memory. *Journal of the ACM*, 40(4):873-890 (1993)

[2] Aguilera M., A pleasant stroll through the land of infinitely many creatures. *ACM SIGACT news, DC column*, 35(2):36-59 (2004)

[3] Attiya H., Gorbach A., and Moran S., Computing in totally anonymous asynchronous shared memory systems. *Information and Computation*, 173(2):162-183 (2002)

[4] Attiya H., Guerraoui R., Hendler D., and Kuznetsov P., The complexity of obstruction-free implementations. *Journal of the ACM*, 56(4), Article 24, 33 pages (2009)

[5] Aspnes J. and Herlihy M., Fast randomized consensus using shared memory. *Journal of Algorithms*, 11:441-461 (1990)

[6] Bonnet F. and Raynal M., Anonymous asynchronous systems: the case of failure detectors. *Distributed Computing*, 26(3):141-158 (2013)

[7] Borowsky E. and Gafni E., Generalized FLP impossibility result for t-resilient asynchronous computations. *Proc. 25-th Annual ACM Symposium on Theory of Computing (STOC'93)*, ACM Press, pp. 91-100 (1993)

[8] Borowsky, E. and Gafni, E. and Lynch, N. and Rajsbaum, S., The BG distributed simulation algorithm. *Distributed Computing*, 14(3):127-146 (2001)

[9] Bouzid Z., Raynal M., and Sutra P., Anonymous obstruction-free (n, k)-set agreement with $n - k + 1$ atomic read/write registers. *Proc. 19th Int'l Conference on Principles of Distributed Systems (OPODIS'15)*, Leibnitz Int'l Proceedings in Informatics (LIPIcs), vol. 46, Article 18:1-17 (2015)

[10] Castañeda A., Rajsbaum S., and Raynal M., Specifying concurrent problems: beyond linearizability and up to tasks. *Proc. 29th Symposium on Distributed Computing (DISC'15)*, Springer LNCS 9363, pp. 420-435 (2015)

[11] Chandra T., Hadzilacos V., and Toueg S., The weakest failure detector for solving consensus. *Journal of the ACM*, 43(4):685-722 (1996)

[12] Chaudhuri S., More *choices* allow more *faults:* set consensus problems in totally asynchronous systems. *Information and Computation,* 105:132-158 (1993)

[13] Delporte C., Fauconnier H., Gafni E., and Rajsbaum S., Black art: obstruction-free k-set agreement with |MWMR registers| $<$ |processes|. *Proc. First Int'l Conference on Networked Systems (NETYS'13)*, Springer LNCS 7853, pp. 28-41 (2013)

[14] Delporte-Gallet C. and Fauconnier H., Two consensus algorithms with atomic registers and failure detector Ω. *Proc. 10th Int'l Conference on Distributed Computing and Networking (ICDCN'09)*, Springer LNCS 5408, pp. 251-262 (2009)

[15] Delporte C., Fauconnier H., Kuznetsov P. and Ruppert E., On the space complexity of set agreement. *Proc. 34th Int'l Symposium on Principles of Distributed Computing (PODC'15)*, ACM Press, pp. 271-280 (2015)

[16] Ellen Fich F., Luchangco V., Moir M., and Shavit N., Obstruction-free algorithms can be practically wait-free. *Proc. 19th Int'l Symposium on Distributed Computing (DISC'05)*, Springer LNCS 3724, pp. 78-92 (2005)

[17] Fischer M.J., Lynch N.A., and Paterson M.S., Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374-382 (1985)

[18] Flocchini P., Prencipe G., Santoro N., and Widmayer P., Hard tasks for weak robots: the role of common knowledge in pattern formation by autonomous mobile robots. *Proc. 10th Int'l Symposium on Algorithms and Computation (ISAAC'99)*, Springer LNCS 1741, pp. 93-102 (1999)

[19] Friedman R., Mostefaoui A., Rajsbaum S., and Raynal M., Distributed agreement problems and their connection with error-correcting codes. *IEEE Transactions on Computers*, 56(7):865-875 (2007)

[20] Gelashvili R., Optimal space complexity of consensus for anonymous processes *Proc. 29th Int'l Symposium on Distributed Computing (DISC'15)*, Springer LNCS 9363, pp. 452-466 (2015)

[21] Guerraoui R. and Ruppert E., Anonymous and fault-tolerant shared-memory computations. *Distributed Computing*, 20:165-177 (2007)

[22] Herlihy M.P., Wait-free synchronization. *ACM Transactions on Programming Languages and Systems*, 13(1):124-149 (1991)

[23] Herlihy M.P., Luchangco V., and Moir M., Obstruction-free synchronization: double-ended queues as an example. *Proc. 23th Int'l IEEE Conference on Distributed Computing Systems (ICDCS'03)*, IEEE Press, pp. 522-529 (2003)

[24] Herlihy M.P. and Shavit N., The topological structure of asynchronous computability.
Journal of the ACM, 46(6):858-923 (1999)

[25] Herlihy M.P. and Shavit N., *The art of multiprocessor programming*. Morgan Kaufmann, 508 pages, ISBN 978-0-12-370591-4 (2008)

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

[27] Lamport L., Concurrent reading and writing. *Communications of the ACM*, 20(11):806-811 (1977)

[28] Lamport L., On interprocess communication, Part I: basic formalism. *Distributed Computing*, 1(2):77-85 (1986)

[29] Loui M.C., and Abu-Amara H.H., Memory requirements for agreement among unreliable asynchronous processes. *Parallel and Distributed Computing: Vol. 4 of Advances in Computing Research,* JAI Press, 4:163-183 (1987)

[30] Merritt M. and Taubenfeld G., Computing with infinitely many processes. *Information & Computation*, 233:12-31 (2013)

[31] Peterson G.L., Concurrent reading while writing. *ACM Transactions on Programming Languages and Systems*, 5:46-55 (1983)

[32] Raynal M., *Concurrent programming: algorithms, principles, and foundations.* Springer, 530 pages, ISBN 978-3-642-32026-2 (2013)

[33] Raynal M. and Stainer J., From the Ω and store-collect building blocks to efficient asynchronous consensus. *Proc. 18th Int'l European Parallel Computing Conference (EUROPAR'12)*, Springer LNCS 7484, pp. 427-438 (2012)

[34] Saks M., Shavit N. and Woll H., Optimal time randomized consensus - making resilient algorithms fast in practice. *Proc. 2nd ACM/SIAM Symposium on Discrete Algorithms (SODA'90)*, ACM Press, pp. 351-362 (1991)

[35] Saks M. and Zaharoglou F., Wait-free k-Set agreement is impossible: the topology of public knowledge. *SIAM Journal Computing* 29(5):1449-1483 (2000)

[36] Suzuki I. and Yamashita M., Distributed anonymous mobile robots. *Proc. 3rd Int'l Colloquium on Structural Information and Communication Complexity (SIROCCO'96)*, Carleton University Press, pp. 313-330 (1996)

[37] Taubenfeld G., *Synchronization algorithms and concurrent programming*. Pearson Education/Prentice Hall, 423 pages, ISBN 0-131-97259-6 (2006)

[38] Taubenfeld G., Contention-sensitive data structure and algorithms. *Proc. 23th Int'l Symposium on Distributed Computing (DISC'09)*, Springer LNCS 5805, pp. 157-171 (2009)

[39] Yamashita M. and Kameda T., Computing on anonymous networks: Part II-decision and membership problems. *IEEE Transactions on Parallel Distributed Systems*, 7(1):90-96 (1996)

[40] Yanagisawa N., Wait-free solvability of colorless takss in anonymous shared-memory model. *Proc. 18th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS'16)*, Springer LNCS 10083, pp. 415-429 (2016)

[41] L Zhu, Brief announcement: tight space bounds for memoryless anonymous consensus. *Proc. 29th Symposium on Distributed Computing (DISC'15)*, Springer LNCS 9363, pp. 665-666 (2015)

[42] L Zhu, A Tight space bound for consensus. *Proc. 48th ACM Symposium on Theory of Computing (STOC'16)*, ACM Press, pp. 345-350 (2016)