Fully Anonymous Consensus and Set Agreement Algorithms*

Michel Raynal[1,2] and Gadi Taubenfeld[3]

[1] Univ Rennes IRISA, France
[2] Department of Computing, Polytechnic University, Hong Kong
[3] The Interdisciplinary Center, Herzliya, Israel

Abstract. Process anonymity has been studied for a long time. Memory anonymity is more recent. In an anonymous memory system, there is no a priori agreement among the processes on the names of the shared registers they access. As an example, a shared register named A by a process p and a shared register named B by another process q may correspond to the very same register X, while the same name C may correspond to different shared registers for the processes p and q. This article focuses on solving the consensus and set agreement problems in the *fully anonymous* model, namely a model in which both the processes and the registers are anonymous. It is shown that consensus, and its weak version called set agreement, can be solved despite full anonymity, in the presence of any number of process crashes. As far as we know, this is the first time where non-trivial concurrency-related problems are solved in such a strong anonymity context. A noteworthy property of the proposed algorithms lies in their conceptual simplicity.

Keywords: Anonymity, Anonymous shared memory, Anonymous processes, Asynchrony, Atomic read/write register, Atomic read/modify/write register, Concurrency, Consensus, Crash failure, Process crash, Set agreement, Obstruction-freedom, Wait-freedom.

1 Introduction: Computing Model

1.1 On the process side

Process anonymity. The notion of *process anonymity* has been studied for a long time from an algorithmic and computability point of view, both in message-passing systems (e.g., [4, 8, 32]) and shared memory systems (e.g., [6, 9, 13]). Process anonymity means that processes have no identity, have the same code and the same initialization of their local variables (otherwise they could be distinguished). Hence, in a process anonymous system, it is impossible to distinguish a process from another process.

Process model. The system is composed of a finite set of $n \geq 2$ asynchronous, anonymous sequential processes denoted p_1, .., p_n. Each process p_i knows n, the number of processes, and m, the number of registers. The subscript i in p_i is only a notational

* A few of the results were mentioned in a brief announcement published in SSS'19 [25].

convenience, which is not known by the processes. *Sequential* means that a process executes one step (instruction) at a time. *Asynchronous* means that each process proceeds in its own speed, which may vary with time and always remains unknown to the other processes. On the failure side, any number of processes may crash (a crash is a premature stop of a process).

1.2 On the memory side

Memory anonymity. The notion of *memory anonymity* has been recently introduced in [30]. Let us consider a shared memory R made up of m atomic registers. Such a memory can be seen as an array with m entries, namely $R[1..m]$. In a non-anonymous memory system, for each index x, the name $R[x]$ denotes the same register whatever the process that accesses the address $R[x]$. Hence in a non-anonymous memory, there is an a priori agreement on the names of the shared registers. This facilitates the implementation of the coordination rules the processes have to follow to progress without violating the safety properties associated with the application they solve [17, 23, 29].

The situation is different in an anonymous memory, where there is no a priori agreement on the name of each register. Moreover, all the registers of an anonymous memory are assumed to be initialized to the same value (otherwise, their initial values could provide information allowing processes to distinguish them). In [24], the interested reader may find an introductory survey on models where (1) only processes are anonymous, and (2) only the memory is anonymous. This paper which considers agreement problems, and [26] which considers the mutual problem, are the first to introduce the notion of fully anonymous shared memory systems, where *both* processes and memory are anonymous.

Anonymous shared memory. The shared memory is made up of $m \geq 1$ atomic anonymous registers denoted $R[1...m]$. Hence, *all* the registers are anonymous. As already indicated, due to its anonymity, $R[x]$ does not necessarily indicate the same object for different processes. More precisely, a memory-anonymous system is such that:

- For each process p_i an adversary defined a permutation $f_i()$ over the set $\{1, 2, \cdots, m\}$, such that when p_i uses the address $R[x]$, it actually accesses $R[f_i(x)]$,
- No process knows the permutations, and
- All the registers are initialized to the same default value denoted $\perp$.

identifiers for an external observer	local identifiers for process p_i	local identifiers for process p_j
$R[1]$	$R_i[2]$	$R_j[3]$
$R[2]$	$R_i[3]$	$R_j[1]$
$R[3]$	$R_i[1]$	$R_j[2]$
permutation	$f_i() : [2, 3, 1]$	$f_j() : [3, 1, 2]$

Table 1. Illustration of an anonymous memory model

An example of anonymous memory is presented in Table 1. To make apparent the fact that $R[x]$ can have a different meaning for different processes, we write $R_i[x]$ when p_i invokes $R[x]$.

Anonymous register model. We consider three types of anonymous register models.

- RW (read/write) model. In this model, all the registers can be read or written by any process.
- RW/Snapshot (in short RW/Snap) model. In this model, all the registers can be read or written by any process. In addition, each process can apply an atomic *snapshot* operation to obtain the values of all the registers in one atomic step. Thus, for example, assuming that processes communicate through a memory anonymous array $R[1..m]$, the operation $R.\texttt{snapshot}()$ obtains the values of all the m entries of the array R in one instantaneous step.[1]
- RMW (read/modify/write) model. In this model, each register can be read, written or accessed by an operation that atomically reads the register and (according to the value read) possibly modifies it. More precisely, this operation denoted $\texttt{compare\&swap}(R[x], old, new)$ has three input parameters, a register $R[x]$ and two values old and new, and returns a Boolean value. It has the following effect: if $R[x] = old$ the value new is assigned to $R[x]$, and the value `true` is returned (the $\texttt{compare\&swap}()$ operation is then successful). If $R[x] \neq old$, $R[x]$ is not modified, and the value `false` is returned.

In these models, *atomic* [19] means that the operations on the registers appear as if they have been executed sequentially, each operation appearing between its start event and its end event, and for any $x \in \{1, ...m\}$, each read operation of a register $R[x]$ returns the value v, where v is the last value written in $R[x]$ by a write or a successful $\texttt{compare\&swap}(R[x], -, -)$ operation (we also say that the execution is *linearizable* [18]). We notice that the RMW model is at least as strong as the RW model.

1.3 Motivation and content of the paper

Motivation. This article addresses consensus and set agreement in fully anonymous systems, and has two primary motivations. The first is related to the basics of computing, namely, computability and complexity lower/upper bounds. Increasing our knowledge of what can (or cannot) be done in the context of full anonymity (i.e., when both the processes and the memory are anonymous), and providing associated necessary and sufficient conditions, helps us determine the weakest system assumptions under which fundamental problems, such as consensus and set agreement can be solved.

The second motivation is application-oriented. It appears that the concept of an anonymous memory allows epigenetic cell modification to be modeled from a computing point of view [27]. In [27] the authors model histone modifiers (which are a specific type of proteins) as two different types of writer processors and two different types of eraser processors that communicate by accessing an anonymous shared memory array which corresponds to a stretch of DNA, and for such a setting formally define the epigenetic consensus problem. Hence, anonymous shared memories could be useful in biologically inspired distributed systems [21, 22]. If this is the case, mastering

[1] For a model where the registers are non-anonymous, it is known that the computational power of the RW model and the RW/Snap model are the same despite asynchrony and any number of process crashes [1, 3]. For fully anonymous systems, this question is open.

agreement problems in such an adversarial context could reveal to be important from an application point of view.

Consensus. Consensus is the most important agreement problem of fault-tolerant distributed computing. Let us consider that any number of processes may crash. A crash is a premature halting (hence, until it possibly crashes, a process behaves correctly, i.e., reliably executes its code). The consensus problem consists in building a one-shot operation, denoted propose(), which takes an input parameter (called *proposed* value) and returns a result (called *decided* value). *One-shot* means that a process can invoke the operation at most once. The meaning of this operation is defined as follows:

- Validity: A decided value is a proposed value.
- Agreement: No two processes decide different values.
- Liveness (Wait-freedom): If a process does not crash, it decides a value.

Algorithms solving consensus in different types of non-anonymous shared memory systems are described in several textbooks (e.g.,[17, 23, 29]). In this paper, we consider the multi-valued version of consensus (i.e., the domain of proposed values is not restricted to be binary). While consensus can be solved from registers in a non-anonymous RMW memory [14], it cannot be solved in a non-anonymous RW memory [12, 20]. It is, however, possible to solve a weaker version of consensus in non-anonymous RW system, when the progress condition is weakened as follows [15]:

- Liveness (Obstruction-freedom): If a process does not crash, and executes alone during a long enough period, it decides. I.e., if a process runs alone starting from some point in the execution then it eventually decides.

Set agreement. Set agreement captures a weaker form of consensus in which the agreement property is weakened as follows:

- At most $n - 1$ different values are decided upon.

That is, in any given run, the size of the set of the decision values is at most $n - 1$. In particular, in runs in which the n processes propose n different values, instead of forcing the processes to agree on a single value, set agreement forces them to eliminate one of the proposed values. The set agreement problem as defined above is also called the $(n - 1)$-set agreement problem [10]. While much weaker than consensus, as consensus, set agreement cannot be solved in crash-prone non-anonymous RW memory systems [7, 16, 28] (and consequently cannot be solved in a crash-prone anonymous memory systems either), but, as consensus, it can be solved when considering the weaker obstruction-freedom progress condition.

Content of the paper. Table 2 describes the technical content of the paper. As an example, the first line associated with set agreement, states that Section 2 presents a set agreement algorithm for an anonymous RW system for any number of $n > 1$ processes and $m \geq 3$ registers.

The paper leaves open the interesting question of whether there exists a fully anonymous obstruction-free consensus algorithm for $n \geq 3$ processes using RW registers.

Problem	Section	Crashes possible?	Register model	Progress condition	# of processes n	# of registers m
Set agreement	2	Yes	RW	Obstruction-freedom	$n > 1$	$m \geq 3$
Consensus	3	Yes	RW	Obstruction-freedom	$n = 2$	$m \geq 3$
Consensus	4	Yes	RW/Snap	Obstruction-freedom	$n > 1$	$m \geq 2n - 1$
Consensus	5	Yes	RMW	Wait-freedom	$n > 1$	$m \geq 1$

Table 2. Structure of the article

2 Fully Anonymous Obstruction-free Set Agreement Using RW Registers

Considering any number $n > 1$ of processes, this section presents an obstruction-free set agreement algorithm for a crash-prone anonymous n-process system, where communication is through $m \geq 3$ anonymous RW registers.

2.1 A fully anonymous RW set agreement algorithm

The algorithm is described in Fig. 1. Each anonymous RW register can store the preference of a process. Each participating process p_i scans the m registers trying to write its preference ($mypref_i$) into each one of the m registers.

Before each write, the process scans the shared array (line 3), and operates as follows:

- If its preference appears in all the m registers (line 8), it decides on its preference and terminates.
- Otherwise, if some preference appears in more than half of the registers (line 4), the process adopts this preference as its new preference (line 5).

Afterward, the process finds some arbitrary entry in the shared array that does not contain its preference (line 6) and writes it into that entry (line 7). Once the process finishes writing it repeats the above steps.

2.2 Proof of the algorithm

Lemma 1 (Set agreement and Termination under Obstruction-freedom). *Any participating process that runs alone for a sufficiently long time, eventually decides. Moreover, the processes that decide, decide on at most $n - 1$ different values.*

Proof. Clearly, in all the runs in which less than n processes decide, they decide on at most $n-1$ different values. So, we have to prove that in runs in which all the n processes participate and decide, the n processes decide on at most $n - 1$ different values.

Let ρ be an arbitrary run in which all the n processes participate and decide. Each one of the n processes, *before* deciding (line 9), must first read all the m registers (line 3), find out that its preference appears in all the m registers (line 8), decide on its

ALGORITHM 1: CODE OF AN ANONYMOUS PROCESS p_i

Constants:
n, m: positive integers // # of processes and # of shared registers
Anonymous RW registers:
$R[1..m]$: array of $m \geq 3$ registers, initially all $\perp$ // $\perp$ cannot be proposed
Local variables:
$myview_i[1..m]$: array of m variables
$mypref_i$: integer; j: ranges over $\{0, ..., m\}$

operation propose(in_i) **is** // in_i value proposed by p_i
1 $mypref_i \leftarrow in_i$
2 **repeat**
3 **for** $j = 1$ **to** m **do** $myview_i[j] \leftarrow R_i[j]$ **od** //read the shared array
4 **if** $\exists\, value \neq \perp$ which appears in more than half of the entries of $myview_i[1..m]$
5 **then** $mypref_i \leftarrow value$ **fi** //update preference
6 $j \leftarrow$ an arbitrary index $k \in \{1, ..., m\}$ such that $myview_i[k] \neq mypref_i$ // search
or 0 if no such index exists
7 **if** $j \neq 0$ **then** $R_i[j] \leftarrow mypref_i$ **fi** // write
8 **until** $\forall j \in \{1, ..., m\} : myview_i[j] = mypref_i$ // my $mypref_i$ is everywhere
9 return$(mypref_i)$. // decide

Fig. 1. Fully anonymous obst.-free set agr. algorithm for $n \geq 2$ proc. and $m \geq 3$ RW registers

preference and terminate. We call this last reads of the m registers by a specific process a *successful collect* (SC) of that process. We emphasize that from the moment a process starts its successful collect until it decides, it does not write.

Let us denote by p_i and p_j the *last* two processes which start their SC in the run ρ. Clearly, by definition, during these two last SCs, each one of the other processes has either decided and terminated or has already started it SC, and hence does not write during p_i and p_j SCs. We show that p_i and p_j must decide on the same value which implies, as required, that the n processes decide on at most $n - 1$ different values in ρ.

From now on we focus only on the processes p_i and p_j. Assume w.l.o.g. that p_i has started its (last and only) SC before process p_j has started its (last and only) SC. Let t_0 and t_1 denote: the last time p_i enters the repeat loop just before reading the m registers (between lines 2-3), and the last time at which p_i exits the repeat loop (between lines 8-9), respectively. At the time interval $[t_0, t_1]$, p_i never writes, and it completes an SC. That is, p_i reads the array once, and finds out that its preference (i.e., $mypref_i$) appears in all the m registers. Let v be the value that p_i reads in its last SC. There are two possible cases.

1. *At time t_0, the values of all the m registers equal v.* After time t_0, and before executing line 3, process p_j might write at most once into one of the m registers possibly overwriting the v value. Thus, when executing line 4, p_j will find that v appears in at least $m - 1$ of the entries of $myview_j[1..m]$. Since $m \geq 3$, this means that p_j will find that v appears in more than half of the entries of $myview_j[1..m]$.

Thus, p_j will set its preference to v (line 5). From that point on, since p_i does not write anymore, the only possible decision value for p_j is v.

2. *At time t_0, not all the values of the m registers equal v.* Since in the time interval $[t_0, t_1]$, p_i has found that the value of each one of the m registers equals v, it must be that process p_j has written the value v into all the registers with values other than v. Thus, p_j when writing v p_j's preference must be v. Since p_i does not write anymore, the only possible decision value for p_j is v.

As both p_i and p_j decide on the same value v in ρ, it follows that the n processes together decide on at most $n-1$ different values in ρ.

Let us now show that, under obstruction-freedom (that is, if it runs alone for a sufficiently long time), each process eventually decides (and terminates). When a process, say process p_i, runs alone from some point on in a computation, p_i will read the shared array (line 3) and set its preference to some value v. From that point on, in each iteration of the repeat loop, p_i will set one more entry of the shared array to v. Thus, after at most m iterations the values of all the m entries will equal v, and p_i will be able to exit the repeat loop, decide v and terminate. □

Lemma 2 (Validity). *The decision value is the input of a participating process.*

Proof. At each point, the current preference of a process is either its initial input or a value (different from $\perp$), it has read from a register. Since a process may only write its preference into a register, the result follows. □

Theorem 1. *Algorithm 1 solves obstruction-free set agreement in a fully anonymous system made up of $n \geq 2$ processes and $m \geq 3$* RW *registers.*

Proof. The proof that the algorithm satisfies the Validity, Agreement, and Obstruction-freedom properties (which define set agreement) follows directly from Lemma 1 and Lemma 2. □

3 Fully Anonymous Obstruction-free Consensus Using RW Registers

As the reader can easily check, instantiating Algorithm 1 with $n = 2$ provides us with 2-process obstruction-free consensus built using $m \geq 3$ *RW* registers.

Corollary 1. *Algorithm 1 solves consensus in a fully anonymous system made up of two processes and $m \geq 3$ anonymous* RW*registers. (In the case of binary consensus, the registers are 3-valued registers.)*

It is interesting to note that while it is possible to solve binary consensus for two processes in a fully anonymous system using only 3-valued registers. It is not possible to do so using only 2-valued registers (i.e., bits). It was recently proved in [31] that there is no obstruction-free consensus algorithm for two non-anonymous processes using only anonymous bits. Thus, as was shown in [31], anonymous bits are strictly weaker than anonymous (and hence also non-anonymous) multi-valued registers.

Let us consider a modified version of Algorithm 1, which assumes $n \geq 3$, in which the requirement $m \geq 3$ is strengthened to $m \geq 2n-1$. It is tempting to think that the resulting algorithm solves obstruction-free consensus for $n \geq 3$ processes.

The (incorrect) supporting argument may go like this. Assume some process p is the first to decide on the value v, after reading that the values of all the $m \geq 2n-1$ registers equal v. Each of the remaining $n-1$ processes, before reading the array (line 3), may write at most once into one of the m registers possibly overwriting the v value. Thus, at most $n-1$ of the values might be overwritten (leaving a majority of v values), before the processes will execute line 3 and find that v appears in more than half of the entries of $myview_i[1..m]$. Each process that finds that v appears in more than half of the entries will set its preference to v (line 5) and must later decide on v.

However, this argument is wrong, and as we prove below the resulting algorithm does not even solve obstruction-free consensus for three processes using five registers.

Theorem 2. *Let $A(n,m)$ be Algorithm 1, in which n is the number of processes and m is the number of anonymous* RW *registers. Then, for any $n \geq 3$ and any $m \geq 1$, $A(n,m)$ does not solve obstruction-free consensus in a fully anonymous system.*

Proof. The proof is by contradiction. Assume $n = 3$ and $m \geq 1$ registers. Clearly, a result for $n = 3$ implies the result for $n \geq 3$. Let us call the processes p_0, p_1, and p_2. Assume that p_0, p_1, p_2 start with inputs 0,1,0, respectively. Furthermore, we prove the result even under the assumption that, in Algorithm 1, $R[1..m]$ is an array of *non-anonymous* registers. So, below we assume that the registers are non-anonymous.

We first build an infinite run, ρ, which involves p_0 and p_1 only, in which the values of each one of the m registers changes from 0 to 1 and vice versa infinitely many times. To this end, we use the function $\mathsf{dist}(a_1, a_2) = (a_2 - a_1) \bmod m$, defined for $a_1, a_2 \in \{1, ..., m\}$. If we think of the m numbers $1, ..., m$ as being arranged clockwise in a circle, then $\mathsf{dist}(a_1, a_2)$ is the distance one must travel clockwise around the circle starting from a_1 before reaching a_2. In the special case that $a_1 = a_2$, the distance is 0. Thus, $0 \leq \mathsf{dist}(a_1, a_2) \leq m-1$, and $a_1 + \mathsf{dist}(a_1, a_2) \equiv a_2 \bmod m$.

For $j \in \{1, ..., m\}$ and $v \in \{0, 1\}$, we define the function $\mathsf{next}(j, v)$ to be the value k such that (1) $R[k] \neq v$, and (2) for every $\ell \in \{1, ..., m\}$ where $R[\ell] \neq v$, $d(k, j) \leq d(\ell, j)$. If we think of the m registers $R[1], ..., R[m]$ as being arranged clockwise in a circle, then $\mathsf{next}(j, v)$ is the closest register to $R[j]$ whose value is different than v, where the distance is measured as the number of steps one must travel clockwise around the circle starting from $R[j]$ before reaching $R[k]$. In the special case that $R[j] \neq v$, $\mathsf{next}(j, v) = j$.

The run ρ, which involves processes p_0 and p_1, is constructed as follows:

$v \to 0$
repeat forever
  for $j = 1$ to m **do**
    $\ell_0 \leftarrow \mathsf{next}(j, 0)$; $\ell_1 \leftarrow \mathsf{next}(j, 1)$
    p_v writes v into $R[\ell_v]$, scans the array and does not change its preference
    p_{1-v} writes $1 - v$ into $R[\ell_{1-v}]$, scans the array and does not change its preference
    $v \leftarrow 1 - v$
  end do
end repeat.

We notice that until all the m registers are written once, in each iteration of the for loop the two processes write into the same register, and thereafter in each iteration they write into different registers. None of the two processes ever needs to change its preference, and each process writes infinitely many times into each one of the registers. Thus, the above procedure produces the required run ρ.

Since the algorithm is only obstruction-free, the existence of such an infinite run is not yet a problem. To produce the counterexample, consider the run ρ. Now let's interleave read operations of the third process p_2 into the run ρ, such that whenever p_2 reads a register it will see the value 0. Thus, at some point, according to the algorithm, p_2 must decide 0 (without ever writing). At that point, let p_1 continue to run alone and, it will decide on 1. A contradiction. □

We point out that (1) this counterexample will not work if the scan of p_2 (reading the m registers) is done in one atomic step (that is using a snapshot operation), and (2) the counterexample applies for the case where the registers are non-anonymous (and hence also for the case where they are anonymous).

It is known that obstruction-free consensus can be solved for n anonymous processes using $O(n)$ non-anonymous RW registers [9, 13]. It is also known that (symmetric) obstruction-free consensus can be solved for n non-anonymous processes using $O(n)$ anonymous RW registers [30]. We leave open the question of whether there exists a *fully anonymous* obstruction-free consensus algorithm for $n \geq 3$ processes using RW registers.

4 Fully Anonymous Obstruction-free Consensus Using RW/Snapshot Registers

For any number $n > 1$ of processes, we present an obstruction-free consensus algorithm for a crash-prone anonymous n-process system, where communication is through $m \geq 2n - 1$ anonymous RW registers which support snapshot operations.

4.1 A fully anonymous consensus algorithm

The algorithm is described in Fig. 2. It is similar to that from Fig. 1, where the scan of the array (line 3) is replaced with a snapshot operation. The anonymous memory is made up of $m \geq 2n - 1$ registers. Each anonymous register can store the preference of a process. Each participating process p_i takes a snapshot of the m registers trying to write its preference ($mypref_i$) into each one of the m registers. Before each write, the process takes a snapshot of the shared array (line 4), and operates as follows:

- If its preference appears in all the m registers (line 8), it decides on its preference and terminates.
- Otherwise, if some preference appears in more than half of the registers (line 4), the process adopts this preference as its new preference (line 5).

Afterward, the process finds some arbitrary entry in the shared array that does not contain its preference (line 6) and writes it into that entry (line 7). Once the process finishes writing it repeats the above steps.

ALGORITHM 2: CODE OF AN ANONYMOUS PROCESS p_i

Constants:
n, m: positive integers // # of processes and # of shared registers
Anonymous RW/Snapshot registers:
$R[1..m]$: array of $m \geq 2n-1$ registers, initially all $\perp$ // $\perp$ cannot be proposed
Local variables:
$myview_i[1..m]$: array of m variables
$mypref_i$: integer; j: ranges over $\{0, ..., m\}$

operation propose(in_i) **is** // in_i value proposed by p_i
1 $mypref_i \leftarrow in_i$
2 **repeat**
3 $\boxed{myview_i[1..m] \leftarrow R.\texttt{snapshot}()}$ //atomic snapshot of the memory
4 **if** $\exists\, value \neq \perp$ which appears in more than half of the entries of $myview_i[1..m]$
5 **then** $mypref_i \leftarrow value$ **fi** //update preference
6 $j \leftarrow$ an arbitrary index $k \in \{1, ..., m\}$ such that $myview_i[k] \neq mypref_i$ // search
or 0 if no such index exists
7 **if** $j \neq 0$ **then** $R_i[j] \leftarrow mypref_i$ **fi** // write
8 **until** $\forall j \in \{1, ..., m\} : myview_i[j] = mypref_i$ // my $mypref_i$ is everywhere
9 return($mypref_i$). // decide

Fig. 2. Fully anony. obst.-free consensus for $n \geq 2$ proc. and $m \geq 2n-1$ RW/Snapshot reg.

4.2 Proof of the algorithm

Lemma 3 (Consensus and Termination under Obstruction-freedom). *Any participating process that runs alone for a sufficiently long time, eventually decides. Moreover, the processes that decide, decide on the same value and terminate.*

Proof. Let process p_i be the first process to decide, and denote the value that p_i decides on by v. This means that, before deciding, after taking a snapshot of the anonymous memory, process p_i has found that, *at a certain moment in time*, the value of each one of the m registers equals v. Each one of the other $n-1$ processes might write into one of the registers overwriting the v value. Since $m \geq 2n-1$, *all* the other processes, when executing line 7, will find that v appears in more than half of the entries of $R[1..m]$ (i.e., v appears in at least $m-n+1$ entries), and each one of them will change its preference to v (line 5). From that point on, the only possible decision value is v.

Let us now show that each process eventually decides (and terminates) under the obstruction-freedom assumption. When a process, say process p_i, runs alone from some point on in a computation, p_i will take a snapshot of the shared array (line 3) and set its preference to v (if it is not v already). From that point on, in each iteration of the repeat loop, process p_i will set one additional entry of the shared array to v. Thus, after at most $m \geq 2n-1$ iterations the values of all the m entries will equal v, and process p_i will be able to exit the repeat loop, decide v and terminate. □

Lemma 4 (Validity). *The decision value is the input of a participating process.*

Proof. At each point, the current preference of a process is either its initial input or a value (different from $\perp$), it has read from a register. Since a process may only write its preference into a register, the result follows. □

Theorem 3. *Algorithm* 2 *solves obstruction-free consensus in a fully anonymous system made up of* $n \geq 2$ *processes and* $m \geq 2n - 1$ RW/Snapshot *registers.*

Proof. The proof that the algorithm satisfies the Validity, Agreement, and Obstruction-freedom properties (which define set agreement) follows directly from Lemma 3 and Lemma 4. □

Remark. Algorithm 1 and Algorithm 2 are actually two instances of an agreement-oriented generic algorithm suited for the crash-prone fully asynchronous model, which ensures termination under the obstruction-freedom assumption. The genericity dimension resides in line 3, which states the way a process reads the content of the anonymous memory, namely an asynchronous scan (Algorithm 1) or a snapshot (Algorithm 2). When $m \geq 2n - 1$ (condition for Algorithm 2), the atomicity of the snapshot operation is powerful enough to go from set-agreement (Algorithm 1) to consensus (Algorithm 2).

5 Fully Anonymous Wait-free Consensus using RMW Registers

When considering a fully anonymous system of size $m = 1$, consensus can be easily solved with the `compare&swap()` operation: the first process that writes its value in the single register $R[1]$ (initialized to $\perp$) imposes it as the decided value (actually, when $m = 1$ the memory is not really anonymous). When using anonymous objects, the fact that a given problem can be solved using only one object (i.e., $m = 1$) does *not* imply that the problem can also be solved using any finite number of $m \geq 1$ objects [5]. For a fully anonymous system, we prove the following simple result,

ALGORITHM 3: CODE OF AN ANONYMOUS PROCESS p_i

Constants:
  n, m: positive integers // `# of processes and # of shared registers`
Anonymous RMW registers:
  $R[1..m]$: array of m RMW registers, initially all $\perp$ // $\perp$ `cannot be proposed`

operation propose(in_i) **is** // in_i `value proposed by` p_i
1 **for each** $j \in \{1, ..., m\}$ **do** `compare&swap`$(R_i[j], \perp, in_i)$ **od** // `try to write`
2 return(max$(R_i[1], ..., R_i[m])$) // `decide the max value in` $R[1..m]$.

Fig. 3. Consensus for $n \geq 2$ anonymous processes and $m \geq 1$ anonymous RMW registers

Theorem 4. *There is a fully anonymous wait-free consensus algorithm for n processes using m* RMW *registers, for any $n \geq 1$ and $m \geq 1$.*

Proof. The simple algorithm described in Fig. 3 presents a simple consensus algorithm for any size $m \geq 1$ of the anonymous RMW memory. This algorithm assumes that the set of values that can be proposed is totally ordered. Each process tries to write the value it proposes into each anonymous register. Assuming that at least one process that does not crash invokes propose(), there is a finite time after which, whatever the concurrency/failure pattern, each anonymous register contains a proposed value. Then, using the same deterministic rule the processes decide the same value (let us notice that there is an a priori statically defined agreement on the deterministic rule used to select the decided value). □

6 Conclusion

This article has several contributions. The first is the introduction, together with [26], of the notion of *fully anonymous* shared memory systems, namely systems where the processes are anonymous, and there is no global agreement on the names of the shared registers (any register can have different names for distinct processes). The article has then addressed the design of agreement algorithms (consensus and set agreement) in specific contexts where the anonymous registers are read/write (RW) registers, RW/snapshot registers, or read/modify/write (RMW) registers. We leave open the interesting question of whether there exists a fully anonymous obstruction-free consensus algorithm for three or more processes using RW registers.

Last but not least, let us notice that, despite the strong adversary context (full anonymity and failures), the proposed algorithms are relatively simple to understand[2]. However, some of their proofs are subtle.

Acknowledgments

M. Raynal was partially supported by the French ANR project DESCARTES (16-CE40-0023-03) devoted to layered and modular structures in distributed computing.

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. Aigner M. and Ziegler G., Proofs from THE BOOK (4th edition). Springer, 274 pages, ISBN 978-3-642-00856-6 (2010)
3. Anderson J.H., Multi-writer composite registers. *Distributed Computing*, 7(4):175-195 (1994)
4. Angluin D., Local and global properties in networks of processes. *Proc. 12th Symposium on Theory of Computing (STOC'80)*, ACM Press, pp. 82-93, (1980)

[2] Let us remind that simplicity is a first class property [2, 11]. A stated by J. Perlis (the recipient of the first Turing Award) "Simplicity does not precede complexity, but follows it".

5. Aghazadeh Z., Imbs D., Raynal M., Taubenfeld G., and Woelfel Ph., Optimal memory-anonymous symmetric deadlock-free mutual exclusion. *Proc. 38th ACM Symposium on Principles of Distributed Computing (PODC'19)*, ACM Press, 10 pages (2019)
6. Attiya H., Gorbach A., and Moran S., Computing in totally anonymous asynchronous shared-memory systems. *Information and Computation*, 173(2):162-183 (2002)
7. Borowsky E. and Gafni E., Generalized FLP impossibility results for t-resilient asynchronous computations. *Proc. 25th ACM Symposium on Theory of Computing (STOC'93)*, ACM Press, pp. 91-100 (1993)
8. Bonnet F. and Raynal M., Anonymous asynchronous systems: the case of failure detectors. *Distributed Computing*, 26(3):141-158 (2013)
9. Bouzid Z., Raynal M., and Sutra P., Anonymous obstruction-free (n, k)-set agreement with $(n - k + 1)$ atomic read/write registers. *Distributed Computing*, 31(2):99-117 (2018)
10. Chaudhuri S., More choices allow more faults: set consensus problems in totally asynchronous systems. *Information and Computation*, 105(1):132-158 (1993)
11. Dijkstra E.W., Some beautiful arguments using mathematical induction. *Algorithmica*, 13(1):1-8 (1980)
12. 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)
13. Guerraoui R. and Ruppert E., Anonymous and fault-tolerant shared-memory computations. *Distributed Computing*, 20:165-177 (2007)
14. Herlihy M.P., Wait-free synchronization. *ACM Transactions on Programming Languages and Systems*, 13(1):124-149 (1991)
15. 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)
16. Herlihy M.P. and Shavit N., The topological structure of asynchronous computability. *Journal ACM*, 46(6):858-923, 1999.
17. Herlihy M. and Shavit N., *The art of multiprocessor programming*. Morgan Kaufmann, 508 pages, ISBN 978-0-12-370591-4 (2008)
18. 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)
19. Lamport L., On interprocess communication, Part I: basic formalism. *Distributed Computing*, 1(2):77-85 (1986)
20. Loui M. and Abu-Amara H., Memory requirements for agreement among unreliable asynchronous processes. *Advances in Computing Research*, 4:163-183, JAI Press (1987)
21. Navlakha S. and Bar-Joseph Z., Algorithms in nature: the convergence of systems biology and computational thinking. *Molecular systems biology*, 7(546):1-11 (2011)
22. Navlakha S. and Bar-Joseph Z., Distributed information processing in biological and computational systems. *Communications of the ACM*, 58(1):94-102 (2015)
23. Raynal M., *Concurrent programming: algorithms, principles and foundations*. Springer, 515 pages, ISBN 978-3-642-32026-2 (2013)
24. Raynal M. and Cao J., Anonymity in distributed read/write systems: an introductory survey. *Proc. 6th Int'l Conference on Networked Systems (NETYS'18)*, Springer LNCS 11028, pp. 122-140 (2018)
25. Raynal M. and Taubenfeld G., Brief Announcement: Fully anonymous shared memory algorithms. *21st International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS'19)*, LNCS 11914, pp. 301-306 (2019)
26. Raynal M. and Taubenfeld G., Mutual exclusion in fully anonymous shared memory systems. *Information Processing Letters*, Volume 158 (June 2020)

27. Rashid S., Taubenfeld G., and Bar-Joseph Z., Genome wide epigenetic modifications as a shared memory consensus problem. *6th Workshop on Biological Distributed Algorithms (BDA'18)*, London (2018)
28. Saks M. and Zaharoglou F., Wait-free k-set agreement is impossible: the topology of public knowledge. *SIAM Journal on Computing*, 29(5):1449-1483 (2000)
29. Taubenfeld G., *Synchronization algorithms and concurrent programming*. Pearson Education/Prentice Hall, 423 pages, ISBN 0-131-97259-6 (2006)
30. Taubenfeld G., Coordination without prior agreement. *Proc. 36th ACM Symposium on Principles of Distributed Computing (PODC'17)*, ACM Press, pp. 325-334 (2017)
31. Taubenfeld G., Set agreement power is not a precise characterization for oblivious deterministic anonymous objects *Proc. 26th International Colloquium on Structural Information and Communication Complexity (SIROCCO19)*, LNCS 11639, pp. 293-308 (2019)
32. Yamashita M. and Kameda T., Computing on anonymous networks: Part I -characterizing the solvable cases. *IEEE Transactions on Parallel Distributed Systems*, 7(1):69-89 (1996)