CHAPTER 7

Valency Arguments

In a valency argument, configurations are classified as either univalent or multivalent. Starting from a *univalent* configuration, all terminating executions (from some class) lead to the same result. In particular, all final configurations are univalent. Starting from a *multivalent* configuration, there are two or more different terminating executions (from the class) that each lead to a different result. When there are only two possible results, for example, in binary consensus, a multivalent configuration is called *bivalent*.

A valency argument typically has two parts. One part is proving that every algorithm has a multivalent initial configuration. This typically follows from the problem specifications. For example, Lemma 2.7 uses a chain argument to prove the existence of a bivalent initial configuration for any binary consensus algorithm.

The other part of a valency argument is proving that, from every multivalent configuration, there is a nonempty execution (from the class) that leads to a multivalent configuration. Together with the existence of a multivalent initial configuration, this implies the existence of an infinite execution containing only multivalent configurations. Since no final configuration is multivalent, the termination property is violated.

Valency arguments were introduced by Michael Fischer, Nancy Lynch, and Michael Paterson in their paper *Impossibility of Distributed Consensus with One Faulty Processor*, Journal of the ACM, volume 32, number 2, 1985, pages 374–382, to show that consensus is impossible in asynchronous message passing systems, even if at most one process can crash.

We begin with a proof that wait-free binary consensus among two or more processes is unsolvable in asynchronous systems where processes communicate via registers. More generally, even if a process can atomically write to a number of registers, wait-free consensus remains unsolvable, if the number of processes is sufficiently large. Then, we give a lower bound on the number of rounds needed to solve binary consensus in synchronous message passing systems. Next, we prove Fischer, Lynch, and Paterson's result, but using a simpler argument. In Section 7.4, we use a valency argument combined with a covering argument to prove a lower bound on the number of registers needed to solve randomized

consensus, starting with the special case when processes are anonymous. Finally, we present a step complexity lower bound for randomized consensus.

7.1 The Impossibility of Consensus using m-assignment

The *consensus number* of an object is the maximum number of processes for which wait-free consensus can be solved in an asynchronous system in which processes communicate using only copies of the object and registers. Thus, proving an upper bound on the consensus number of an object is equivalent to proving that consensus cannot be solved using only copies of the object and registers when the number of processes is sufficiently large.

The model we consider is asynchronous shared memory, where processes communicate via m-assignment objects. An m-assignment object consists of an array of registers that can be read one at a time and any m (or less) of which can be written to simultaneously:

$$r_{i_1}, \ldots, r_{i_m} \leftarrow v_1, \ldots, v_m.$$

When $m = 1$, processes write to one register at a time. In other words, a 1-assignment object is simply an array of registers that each support read and write.

A *critical configuration* is a multivalent configuration where one step by any process moves the system into a univalent configuration. We prove, by contradiction, that critical configurations cannot exist when there are only m-assignment objects. Different cases are considered, depending on what step each process is about to take. In each case, we identify two univalent configurations from which different values are decided. We also identify a set of processes that cannot distinguish between these configurations and a part of the environment that is the same in both configurations. Next, we choose a terminating execution by processes in this set that only accesses this part of the environment. Then, by Corollary 2.2, there is an execution starting from the second configuration which is indistinguishable from the first execution to all processes in the set. Thus, these processes decide the same value in both executions and a contradiction is obtained.

Theorem 7.1. *It is impossible to solve wait-free binary consensus among n processes that communicate using m-assignment objects if $n \geq 2$ and $m = 1$ or $n \geq 2m - 1$ and $m \geq 2$.*

Proof. Suppose there is an implementation of wait-free binary consensus among n processes, where $m = 1$ and $n \geq 2$ or $m \geq 2$ and $n \geq 2m - 1 \geq 3$. Consider any critical configuration C. Let P_0 be the set of processes whose step from C takes the system to a univalent configuration from which only 0 is decided and let P_1 be the set of processes whose step from C takes the system to a univalent configuration from which only 1 is decided. Then P_0 and P_1 partition the set $\{p_0, \ldots, p_{n-1}\}$ of all processes. Since C is bivalent, neither P_0 nor P_1 is empty.

Suppose that some process p_i does a read starting from C. Let p_j be a process in the other part of the partition. Let C' be the configuration obtained from C by performing

one step of p_j. Let C'' be the configuration obtained from C by performing one step of p_i and then one step of p_j. Since p_i and p_j are in different parts of the partition, different values are decided starting from C' and C''. However, $C' \overset{p_j}{\sim} C''$ and all registers have the same values in C' and C'', so p_j decides the same value in solo terminating executions starting from C' and C''. This is a contradiction.

Hence, in configuration C, each process is about to write to at most m different registers. Let R_i denote the set of at most m registers that p_i covers in C.

In configuration C, each process p_i covers some register that is not covered by any other process, i.e. $R_i \not\subseteq \cup\{R_k \mid k \neq i\}$. To see why, suppose that $R_i \subseteq \cup\{R_k \mid k \neq i\}$. Consider any process p_j in the other part of the partition. Let α be a history that consists of one step of p_j, in which it writes to the registers in R_j, followed by one step by p_k, for each $k \neq i, j$, in which it writes to the registers in R_k. Let σ be the history that consists of one step by p_i, in which it writes the registers in R_i. Since p_i and p_j are in different parts of the partition, different values are decided starting from $C\alpha$ and $C\sigma\alpha$. However, $C\alpha \overset{p_j}{\sim} C\sigma\alpha$. Furthermore, all registers have the same values in $C\alpha$ and $C\sigma\alpha$, since all the registers to which p_i writes are overwritten during α. Therefore p_j decides the same value in solo terminating executions starting from $C\alpha$ and $C\sigma\alpha$. This is a contradiction.

Next, we show that, in configuration C, for each process $p_i \in P_0$ and each process $p_j \in P_1$, there is some register that is covered by both p_i and p_j, but is not covered by any other process, i.e. $R_i \cap R_j \not\subseteq \cup\{R_k \mid k \neq i, j\}$. To see why, suppose that $R_i \cap R_j \subseteq \cup\{R_k \mid k \neq i, j\}$. Let σ be the history that consists of one step of p_i, in which it writes to each register in R_i, and let τ be the history starting from C that consists of one step of p_j, in which it writes to each register in R_j. Then all executions starting from $C\sigma$ decide 0 and all executions starting from $C\tau$ decide 1. Let α be a history that consists of one step of every process p_k, for $k \neq i, j$. Since $R_i \cap R_j \subseteq \cup\{R_k \mid k \neq i, j\}$, every register written during both σ and τ is overwritten during α. Hence configurations $C\sigma\tau\alpha$ and $C\tau\sigma\alpha$ are identical. Thus, all terminating executions starting from $C\tau\sigma\alpha$ and $C\sigma\tau\alpha$ decide the same value. This is a contradiction.

Without loss of generality, suppose that $|P_1| \geq |P_0|$. Let p_i be a process in P_0. Then R_i contains at least $|P_1| + 1$ registers: one that is covered by p_i, but by no other process, and, for each $p_j \in P_1$, one that is covered by p_i and p_j, but by no other process. Since $|P_0 \cup P_1| = n$, it follows that $|P_1| \geq \lceil n/2 \rceil \geq m$ and, thus, $|R_i| \geq m + 1$. This is a contradiction, because $|R_i| \leq m$. □

This result is due to Maurice Herlihy in his paper *Wait-free Synchronization*, ACM Transactions on Programming Languages and Systems, volume 13, number 1, 1991, pages 124-149. That paper also gives a matching upper bound: a wait-free consensus algorithm for $2m - 2$ processes. Thus, the consensus number of the m-assignment object is $2m - 2$ and there are objects with arbitrarily large consensus number.

7.2 An $f + 1$ Round Lower Bound for Consensus

Valency arguments can also be used to obtain lower bounds on the number of rounds to solve consensus in synchronous models. Here, we consider the synchronous message passing model defined in Section 2.4 and give another proof of Theorem 2.9. Recall that we assume at most f processes crash in each execution. There is a similar proof for the shared memory model in which processes communicate via registers.

Theorem 7.2. *Any binary consensus algorithm in a synchronous message passing system with $n \geq f + 2$ processes that tolerates f crashes requires more than f rounds, even if at most one process crashes in each round.*

Proof. We restrict attention to the class of synchronous executions in which at most one process crashes in each round. It suffices to prove that, starting from any bivalent configuration in which fewer than f processes have crashed, there is a round in which at most one process crashes and which results in a bivalent configuration. Essentially, we show that the adversary may crash one process during each round to maintain multivalence. This can continue as long as there remains enough processes for the adversary to crash.

Let C be a bivalent configuration in which fewer than f processes have crashed. Since $n \geq f + 2$, there are at least three processes that have not crashed. To obtain a contradiction, suppose that each round starting from C in which at most one process crashes results in a univalent configuration. Let α be the round starting from C in which no process crashes and let v be the value decided by all terminating executions in the class that start from configuration $C\alpha$. Since C is bivalent, the set of rounds starting from C in which one process crashes and from which all terminating executions in the class decide $1 - v$ is nonempty. Among all the rounds in this set, let β be one in which the process p_i that crashes sends the largest number of messages.

First, suppose that, in round β, process p_i sends a message to every other process p_j that has not crashed in C. Then $C\alpha \overset{p_j}{\sim} C\beta$. Consider any terminating execution by the remaining processes starting from $C\alpha$ that is in the class. It decides v. By Lemma 2.1, the same sequence of events can occur starting from $C\beta$. This is impossible, since every execution in the class starting from $C\beta$ decides $1 - v$.

Therefore, there is some process p_k that has not crashed in C to which p_i does not send a message in β. Let γ be the round starting from C that is the same as β except that p_i also sends a message to p_k. Then $C\gamma \overset{p_j}{\sim} C\beta$ for all processes $p_j \neq p_i, p_k$ that have not crashed in C. Consider any terminating execution by these processes starting from $C\gamma$ that is in the class. By definition of β, this execution decides v. However, by Lemma 2.1, the same sequence of events can occur starting from $C\beta$. This is impossible, since every execution in the class starting from $C\beta$ decides $1 - v$. □

This proof is from Macros Aguilera and Sam Toueg, *A simple bivalency proof that t-resilient consensus requires $t + 1$ rounds*, Information Processing Letters, volume 71, 1999, pages 155-158.

7.3 The Impossibility of Consensus in Asynchronous Message Passing Systems

A similar proof can be used to prove that consensus is unsolvable in asynchronous message passing systems. Instead of crashing a process each round, it suffices to delay a process each round. Thus, the adversary never runs out of processes.

Theorem 7.3. *It is impossible to solve wait-free binary consensus in an asynchronous message passing system in which one process can crash.*

Proof. We restrict attention to a class A of almost synchronous executions, each of which consists of a sequence of rounds. In each round, all or all but one of the processes each takes a step, in which it sends a message to every other process and then receives messages that have been sent to it. There are three types of rounds: In a *full* round, every process takes exactly one step in which it sends a message to every other process and then receives all messages that have been sent to it, but not yet received, including those sent during the round by processes that were scheduled earlier. A *partial* round is the same as a full round, except that one process does not take a step. Finally, a p_i-round is like a full round, except that process p_i is not scheduled first, and it does not receive the message sent to it during the round by the process scheduled immediately before it.

It suffices to prove that, starting from any bivalent configuration at the end of a round, there is a round which results in a bivalent configuration. Let C be a bivalent configuration at the end of some round. To obtain a contradiction, suppose that every round starting from C results in a univalent configuration.

Consider any full round α and let β be the partial round obtained from α by removing the step by its last process, p_i. Let α' be any full round starting with a step by process p_i and let β' be the partial round obtained from α' by removing its first step. Then $C\alpha\beta'$ and $C\beta\alpha'$ are indistinguishable to all processes, since they consist of the same sequence of steps. Hence, any terminating execution in A starting from these two configurations must decide the same value. Since $C\alpha$ and $C\beta$ are both univalent, it follows that all terminating executions in A starting from them must decide this same value, as well.

Next, consider any p_i-round γ and suppose that process p_j is scheduled immediately before p_i. Let γ' be the p_j-round execution obtained from γ by interchanging the order of p_i and p_j. Since neither p_i nor p_j receives the message sent to one another during this round, configurations $C\gamma$ and $C\gamma'$ are indistinguishable to all processes. Hence, all terminating executions in A starting from these two univalent configurations must decide the same value.

Now, let δ be the full round in which processes take steps in the same order as in γ. Since p_i sends all its messages before receiving any messages, $C\gamma$ and $C\delta$ are indistinguishable to all processes except p_i. Then any terminating execution of partial rounds starting from these two configurations in which p_i takes no steps (i.e. in which p_i has crashed) must decide the same value. Hence, all terminating executions in A starting from these two univalent configurations must decide the same value.

Finally, suppose there are two rounds α_0 and α_1 such that all terminating executions in A starting from $C\alpha_0$ decide 0 and all terminating executions in A starting from $C\alpha_1$ decide 1. Without loss of generality, we may assume that α_0 and α_1 are both full rounds. Since any permutation of $\{p_1, \ldots, p_n\}$ can be converted into any other by a sequence of transpositions, we may assume that α and α_1 are identical, except that some process p_i is scheduled immediately before p_j in α_0 and immediately after p_j in α_1. Let γ_0 be the p_j-round in which processes take steps in the same order as in α and let γ_1 be the p_i-round in which processes take steps in the same order as in α_1. Then, all terminating executions in A starting from $C\gamma_0$ decide 0 and all terminating executions in A starting from $C\gamma_1$ decide 1. This contradicts the fact that all terminating executions in A starting from these two configurations must decide the same value.

Thus, all terminating executions starting from C decide the same value. This contradicts the assumption that C is bivalent. □

The proof of this lower bound was adapted from Yoram Moses and Sergio Rajsbaum, *A Layered Analysis of Consensus*, SIAM J. Comput., vol. 31, no. 4, pages 989–1021. Essentially the same proof can also be used for shared memory models in which processes communicate using single-writer registers or a single-writer snapshot object.

7.4 A Space Lower Bound for Consensus

From Section 7.1, we know that in an asynchronous shared memory model where processes only communicate via registers, wait-free consensus is impossible. However, consensus is possible with weaker termination conditions. There are randomized algorithms which terminate in finite expected time. There are deterministic obstruction-free algorithms, in which a process terminates if it is given sufficiently many consecutive steps. The termination condition we use in this section, *nondeterministic solo termination*, is weaker than both of these. It requires that, from every configuration and for all processes, p, there is a finite solo execution by p in which p terminates.

Even though we are considering weaker termination conditions, the outputs of all executions must still satisfy the agreement and validity properties. To show that an algorithm is faulty, we construct an execution that decides both 0 and 1. For example, suppose there is a reachable configuration C in which all the registers are covered by a set of processes P and there is a solo execution α starting from C by a process $q \notin P$ that decides 1. Consider the execution of Figure 7.1, in which the processes in P perform a block write β and then a process $p \in P$ performs a solo execution γ starting from the resulting configuration. If p decides 0, then $\alpha\beta\gamma$ is an execution that decides both 0 and 1.

We use a combination of valency and covering arguments to prove an $\Omega(\sqrt{n})$ lower bound on the number of registers used by any consensus algorithm that satisfies nondeterministic solo termination. Specifically, we show that, if the number of processes is sufficiently large relative to the number of registers, then it is possible to construct an execution that decides both 0 and 1. Starting from any bivalent configuration, we either

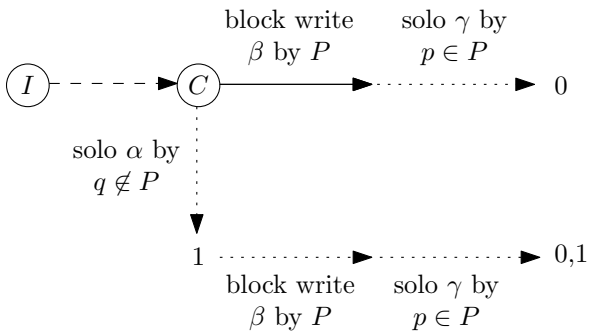

Figure 7.1. A simple situation where agreement can be violated

construct an execution that decides both 0 and 1, or prove that it is possible to reach a bivalent configuration in which more registers are covered. This gives an upper bound on the number of processes (as a function of the number of registers) in any correct algorithm satisfying nondeterministic solo termination, which implies a lower bound on the number of registers (as a function of the number of processes), and also for randomized algorithms and deterministic obstruction-free algorithms.

Anonymous Processes

We begin by proving the lower bound in a system of *anonymous* processes. This means that all processes are identical and they run the same code. If two such processes are in the same state, they apply the same primitive to the same object when they are next allocated a step and, if the results are the same (for example, they read the same value or they get the same outcome from a coin flip), then they go to the same state. Initially, all processes with the same input value will be in the same state. Although this model is quite restrictive, it provides important insight for the lower bound in the general case.

A *clone* of a process p is a process with the same input as p, which proceeds in lockstep with p, reading and writing the same values as p, until immediately before some write to a register. An adversary can have the clone apply that write at some later point in the execution to ensure that the value p reads from that register is the same as the value that p last wrote there. After applying its delayed write, a clone takes no further steps. Note that, until a clone does its delayed write, other processes, including p, are unaware of its existence. In other words, the execution with this clone and without it are indistinguishable to them.

Let r denote the number of registers. The following lemma shows that, if there is a bivalent configuration in which there are sufficiently many processes available to be used as clones, then it is possible to construct an execution that decides both 0 and 1. The situation is illustrated in Figure 7.2.

Lemma 7.4. *Consider a reachable configuration C in which there is a set of processes P covering a set of registers V, a disjoint set of processes Q covering a (not necessarily*

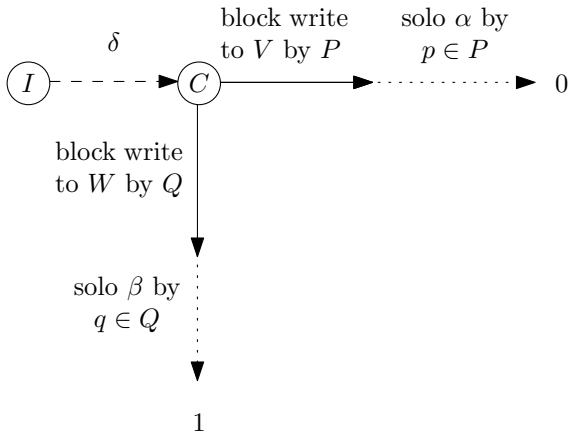

Figure 7.2. The situation in Lemma 7.4

disjoint) set of registers W, and at least $r^2 - r + (|V| + |W| - |V|^2 - |W|^2)/2$ processes that have taken no steps and are not in $P \cup Q$. Suppose that after the block write *by P there is a solo execution α by a process $p \in P$ in which p decides 0 and after the* block write *by Q there is a solo execution β by a process $q \in Q$ in which q decides 1. Then there is an execution that decides both 0 and 1.*

Proof. By induction on (V, W), where pairs of sets are partially ordered by component-wise inclusion and the base case of the induction is when at least one of these two sets consists of all the registers (i.e. $|V| = r$ or $|W| = r$). Consider any pair (V, W) and suppose that the claim is true for all $(V', W') \neq (V, W)$ such that $V \subseteq V'$ and $W \subseteq W'$. We consider two cases.

Case 1. $V \subseteq W$. The case $W \subseteq V$ is symmetric.

First, assume all writes that occur during α are to registers in W. For example, this happens in the base case, when $|W| = r$. Consider the execution to C followed by the block write to V by P, α, the block write to W by Q, and β. This execution, illustrated in Figure 7.3, decides both 0 and 1.

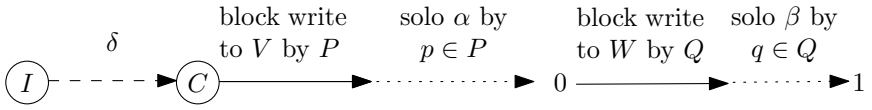

Figure 7.3. The case when $V \subseteq W$ and all writes during α are to registers in W.

Otherwise, let α' be the longest prefix of α that only contains writes to registers in W, let $R \notin W$ be the register covered by p immediately after α', and let α'' be the remainder of α, following the write by p to R. Let C' be the configuration immediately after α', except that there is a clone covering each register in V, which was left behind

when that register was last written to. The block write to V by the set, P', of these clones starting from configuration C' does not change the values of any registers, so the solo execution by p consisting of a write to R followed by α'' can be performed starting from the resulting configuration. When the processes in Q perform a block write starting from C' instead of from C, the resulting configurations are indistinguishable to process q, so the solo execution β by q still decides 1. Therefore, we have the situation depicted in Figure 7.4.

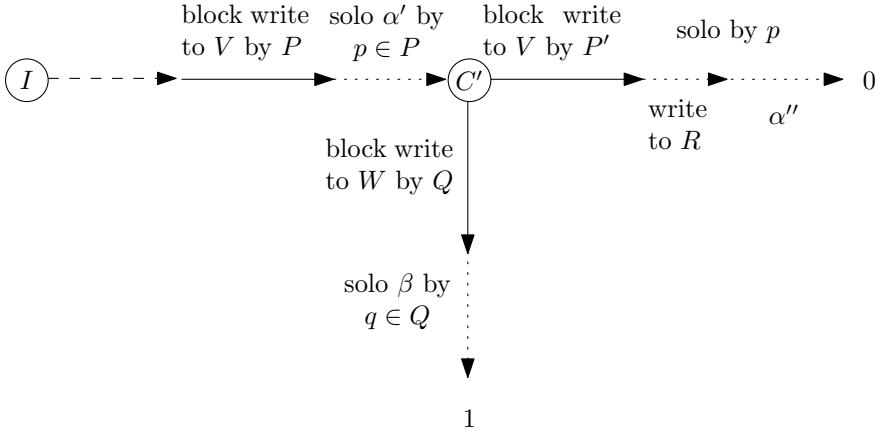

Figure 7.4. The case when $V \subseteq W$ and α contains a write to a register $R \notin W$.

Let $V' = V \cup \{R\}$, so $|V'| = |V| + 1$. In C', the number of processes that have taken no steps and are not in $P' \cup Q$ is at least

$$r^2 - r + (|V| + |W| - |V|^2 - |W|^2)/2 - |V| = r^2 - r + (|V'| + |W| - |V'|^2 - |W|^2)/2.$$

By the induction hypothesis for (V', W), with C' instead of C and $P' \cup \{p\}$ instead of P, there is an execution that decides both 0 and 1.

Case 2. $V \not\subseteq W$ and $W \not\subseteq V$. Let $U = V \cup W$. Then $V, W \subsetneq U$. Consider any terminating execution starting from C that begins with a block write to U and continues with a solo execution γ by one of these processes. The existence of γ is guaranteed by nondeterministic solo termination. Without loss of generality, suppose that γ decides 0. Let C' be the configuration that is the same as C, except there is a clone covering each register in $W - V = U - V$, which was left behind when that register was last written to. Let P' be a set of processes disjoint from Q covering U in C'. This is illustrated in Figure 7.5.

In C', the number of processes that have taken no steps and are not in $P' \cup Q$ is at

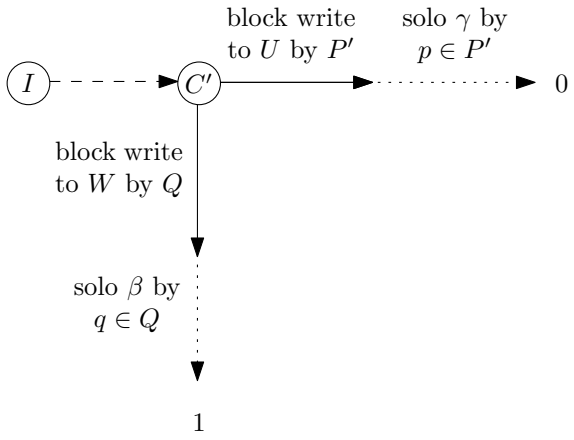

Figure 7.5. The case when $V \not\subseteq W$ and $W \not\subseteq V$.

least

$$\begin{aligned}
&r^2 - r + (|V| + |W| - |V|^2 - |W|^2)/2 - |W - V| \\
&\quad = r^2 - r + (|U| - |W - V| + |W| - (|U| - |W - V|)^2 - |W|^2 - 2|W - V|)/2 \\
&\quad = r^2 - r + (|U| + |W| - |U|^2 - |W|^2)/2 + |W - V| \cdot (|V| - |W - V|/2 - 3/2) \\
&\quad \geq r^2 - r + (|U| + |W| - |U|^2 - |W|^2)/2,
\end{aligned}$$

since $|V| \geq 1$ and $|W - V| \geq 1$. By the induction hypothesis for (U, W), with C' instead of C and P' instead of P, there is an execution that decides both 0 and 1. □

Using this lemma, we show that no consensus algorithm exists, if the number of anonymous processes is sufficiently large compared to the number of registers.

Theorem 7.5. *There is no consensus algorithm using r registers for $r^2 - r + 2$ or more processes.*

Proof. Suppose there is such an algorithm. Let C_0 be an initial configuration in which p has input 0 and q has input 1. Then there is a solo execution α from C_0 by p that decides 0 and a solo execution β from C_0 by q that decides 1. If p doesn't write during α, then $\alpha\beta$ is the desired execution and if q doesn't write during β, then $\beta\alpha$ is the desired execution. So suppose that p first writes to R and q first writes to R'. Say that $\alpha = \alpha'\alpha''$, where α' is the longest prefix of α that contains no writes and $\beta = \beta'\beta''$, where β' is the longest prefix of β that contains no writes. Let $P = \{p\}$, $Q = \{q\}$, $V = \{R\}$, $W = \{R'\}$, and $C = C_0\alpha'\beta'$. In configuration C, the number of processes that have taken no steps and are not in $P \cup Q$ is at least $r^2 - r = r^2 - r + (|V| + |W| - |V|^2 - |W|^2)/2$. Therefore, by Lemma 7.4, there is an execution that decides both 0 and 1. This contradicts the correctness of the algorithm. □

The General Case

When processes are not anonymous, it is more difficult for the adversary to get multiple processes to cover the same register. Although the structure of the proof is the same as in the case of anonymous processes, there is more bookkeeping and combinatorics involved. The following notation is helpful. For any set of registers V, let $\overline{V}$ denote the set of registers not in V. Then $|\overline{V}| = r - |V|$, where r is the number of registers in the system.

The key to the lower bound is the following definition. It is the analogue of a terminating solo execution with added clones.

Definition 1. Let P be a set of processes and let V be a set of registers. An execution $\alpha = \alpha_1\alpha'$ starting from configuration C is *interruptible for P and V* if

- in C, there are at least $|\overline{V}| + 1$ processes in P covering every register in V,
- α_1 begins with a block write to V,
- all writes in α_1 are to registers in V,
- all steps of α are by processes in P,
- some process in P decides by the end of α, and
- either $\alpha = \alpha_1$ or there exist a set of processes $P' \subseteq P$ and a set of registers $V' \supsetneq V$ such that α' is interruptible for P' and V'.

Note that, if an execution is interruptible for P and V, it is also interruptible for P'' and V, for all $P'' \supseteq P$. It may be helpful to consider the following equivalent, noninductive definition: An execution α starting from configuration C is interruptible for a set of processes P and a set of registers V if α can be divided into one or more pieces $\alpha = \alpha_1 \cdots \alpha_k$ and there exist $V = V_1 \subsetneq V_2 \subsetneq \cdots \subsetneq V_k$ and $P = P_1 \supseteq P_2 \supseteq \cdots \supseteq P_k$ such that, for $i = 1, \ldots, k$,

- α_i begins with a block write to a set of registers V_i,
- all writes in α_i are to registers in V_i,
- at the beginning of α_i (i.e. immediately after $\alpha_1 \cdots \alpha_{i-1}$, if $i > 1$) there are at least $|\overline{V_i}| + 1$ processes in P_i covering every register in V_i,
- all steps of α_i are by processes in P_i, and
- some process in P_k decides by the end of $\alpha = \alpha_1 \cdots \alpha_k$.

After each piece of an interruptible execution, there are more registers covered, but there may be fewer processes covering a particular register. This execution is interruptible in the sense that it is possible to insert certain executions by processes not in P between the pieces of α so that the resulting execution is indistinguishable from α to the processes

in P. Specifically, let $\beta_1, \ldots, \beta_k, \beta_{k+1}$ be executions by processes not in P such that for $i = 1, \ldots, k$, all writes in β_i are to registers in V_i. Then $\alpha \overset{P}{\sim} \beta_1\alpha_1 \cdots \beta_k\alpha_k\beta_{k+1}$.

While it is straightforward to add clones to an execution when processes are anonymous, the existence of an interruptible execution for P and V starting from configuration C requires a careful proof, which depends on P being sufficiently large and, in configuration C, each register in V being covered by sufficiently many processes in P. We use the following simple combinatorial fact.

Proposition 7.6. *If $x_1 \geq \cdots x_k$ is a sequence of integers such that $\sum_{i=1}^{k} x_i > k(k-1)/2$, then there exists $i \in \{1, \ldots, k\}$ such that $x_i \geq k + 1 - i$.*

Proof. Suppose not. Then $x_i \leq k - i$ for $i = 1, \ldots, k$ and $\sum_{i=1}^{k} x_i \leq \sum_{i=1}^{k} k - i = \sum_{j=0}^{k-1} j = k(k-1)/2$, which contradicts the assumption. □

Using a simplified version of Lemma 7.7 (with $Y = \phi$) we can show that, if $|P| > (r^2 - r + |V| - |V|^2)/2$ and, in configuration C, there are at least $|\overline{V}| + 1$ processes in P covering every register in V, then there is an interruptible execution for P and V starting from C. However, this is insufficient to get an analogue of Lemma 7.4 without using clones. Specifically, in the proof of the second case of Lemma 7.4, clones of processes in P that are covering registers in V are combined with clones of processes in Q covering registers in W to create a new execution. To facilitate this, we introduce the concept of reserving processes for a set of registers.

Definition 2. An interruptible execution $\alpha = \alpha_1\alpha'$ for P and V starting at configuration C *reserves processes for* a set of registers Y if there are at least $|Y|$ processes not in P covering every register in $Y \cap V$ in configuration C and either $\alpha = \alpha_1$ or α' is an interruptible execution for $P' \subseteq P$ and $V' \supsetneq V$ that reserves processes for Y.

Note that, if α reserves processes for Y and $Y' \subseteq Y$, then α reserves processes for Y'.

Lemma 7.7. *Suppose that, in configuration C, there are at least $|\overline{V}| + 1$ processes in P covering every register in V and there are at least $|Y|$ processes not in P covering every register in $V \cap Y$. If $|P| > |Y| \cdot |Y \cap \overline{V}| + (r^2 - r + |V| - |V|^2)/2$, then there is an interruptible execution for P and V starting from C that reserves processes for Y.*

Proof. By induction on $|\overline{V}|$.

Let $\widehat{P} \subseteq P$ contain $|\overline{V}|$ processes covering each register in V. Then $|P - \widehat{P}| = |P| - |\overline{V}| \cdot |V|$. Let α_1 be an execution starting from C that begins with a block write to V by processes in $P - \widehat{P}$ and, one at a time, each process in $P - \widehat{P}$ takes steps until it is covering a register in $\overline{V}$ or it decides, whichever happens first. Nondeterministic solo termination guarantees the existence of such an execution.

If some process in P decides in α_1, then α_1 is an interruptible execution for P and V that reserves processes for Y. In particular, if $|\overline{V}| = 0$, then every process in $P - \widehat{P}$ decides in α_1.

Now suppose $|\overline{V}| \geq 1$ and, in the configuration C' at the end of α_1, every process in $P - \widehat{P}$ is covering a register in $\overline{V}$. For each register $R \in \overline{V}$, let $x'(R)$ denote the number of

processes in $P - \widehat{P}$ covering R in configuration C'. Let $x(R) = x'(R) - |Y|$ if $R \in Y \cap \overline{V}$, let $x(R) = x'(R)$ if $R \in \overline{Y} \cap \overline{V}$, and let $x_1 \geq x_2 \geq \cdots \geq x_{|\overline{V}|}$ be a sorted list of the numbers $x(R)$, for $R \in \overline{V}$. Then

$$\begin{aligned}\sum_{i=1}^{|\overline{V}|} x_i &= \sum_{R \in \overline{V}} x'(R) - |Y| \cdot |Y \cap \overline{V}| \\ &= |P - \widehat{P}| - |Y| \cdot |Y \cap \overline{V}| \\ &= |P| - |\overline{V}| \cdot |V| - |Y| \cdot |\overline{V} \cap Y| \\ &> (r^2 - r + |V| - |V|^2)/2 - |\overline{V}| \cdot |V| \\ &= |\overline{V}|(|\overline{V}| - 1)/2.\end{aligned}$$

By Lemma 7.6, there exists $i \in \{1, \ldots, |\overline{V}|\}$ such that $x_i \geq |\overline{V}| + 1 - i$. Thus, there is a set $S \subseteq \overline{V}$ of i registers such that $x(R) \geq |\overline{V}| + 1 - i$ for all $R \in S$ and let $V' = V \cup S$. Let P' be obtained from P by removing $|Y|$ processes covering each register in $Y \cap S$. Since only processes in P take steps in α_1, there are at least $|Y|$ processes not in P and, hence not in P', covering each register in $Y \cap V$. Thus, there are at least $|Y|$ processes not in P' covering each register in $Y \cap V'$.

There are $x'(R) - |Y| = x(R)$ processes in P' covering each register in $Y \cap S$ and $x'(R) = x(R)$ processes in P' covering each register in $\overline{Y} \cap S$. Thus, there are $x(R) \geq |\overline{V}| + 1 - i = |\overline{V'}| + 1$ processes in P' covering each register in S. There are also $|\overline{V}| \geq |\overline{V'}| + 1$ processes in $\widehat{P} \subseteq P'$ covering each register in V. Hence, there are at least $|\overline{V'}| + 1$ processes in P' covering each register in V'.

Since $|V'| > |V| \geq 1$ and $f(v) = v - v^2$ is a nonincreasing function of the non-negative integers, it follows that

$$\begin{aligned}|P'| &= |P| - |Y| \cdot |Y \cap S| \\ &> |Y| \cdot |Y \cap \overline{V}| + (r^2 - r - |V|^2 + |V|)/2 - |Y| \cdot |Y \cap S| \\ &= |Y| \cdot |Y \cap \overline{V'}| + (r^2 - r - |V|^2 + |V|)/2 \\ &\geq |Y| \cdot |Y \cap \overline{V'}| + (r^2 - r - |V'|^2 + |V'|)/2\end{aligned}$$

So, by the induction hypothesis, there is an interruptible execution α' for P' and V starting from C' that reserves processes for Y. Hence $\alpha_1\alpha'$ is an interruptible execution for P and V starting from C that reserves behind processes for Y. □

Now we can prove a lemma similar to Lemma 7.4, provided P and Q are sufficiently large.

Lemma 7.8. *Let P and Q be disjoint sets of processes and let V and W be (not necessarily disjoint) sets of registers. Consider a configuration C from which there is an interruptible execution $\alpha = \alpha_1\alpha'$ for P and V that decides 0 and reserves processes for $\overline{W}$ and an interruptible execution β for Q and W that decides 1 and reserves processes for $\overline{V}$. If $|P| \geq |\overline{W}| \cdot |\overline{W} \cap \overline{V}| + (r^2 - r + |V| - |V|^2)/2$ and $|Q| \geq |\overline{V}| \cdot |\overline{V} \cap \overline{W}| + (r^2 - r + |W| - |W|^2)/2$, then there is an execution starting from C that decides both 0 and 1.*

Proof. By induction on (V, W), as in the proof of Lemma 7.4. Consider any pair (V, W) and suppose that the lemma is true for all $(V', W') \neq (V, W)$ such that $V \subseteq V'$ and $W \subseteq W'$. We consider two cases.

Case 1. $V \subseteq W$. The case $W \subseteq V$ is symmetric.

First suppose that all writes that occur during α are to registers in W. For example, this happens in the base case of the induction, when $|W| = r$. Then $\alpha\beta$ starting from C decides both 0 and 1.

Otherwise, α' is interruptible for some $P' \subseteq P$ and $V' \supsetneq V$ and reserves processes for $\overline{W}$. Let C' be the configuration at the end of α_1. Since all writes in α_1 are to registers in V and β begins with a block write to $W \supseteq V$, the executions β and $\alpha_1\beta$ starting from C are indistinguishable to processes in Q. Furthermore, $\overline{V'} \subseteq \overline{V}$. Therefore β is interruptible for Q and W starting from C' and leaves behind processes for $\overline{V'}$.

Consider configuration C'. Since α' reserves processes for $\overline{W}$, there are at least $|\overline{W}|$ processes not in P' covering each register in $\overline{W} \cap V'$. Let P'' be a maximum size set such that $P' \subseteq P'' \subseteq P$ and there are at least $|\overline{W}|$ processes not in P'' covering each register in $\overline{W} \cap V'$. Then α' is also interruptible for P'' and V' and reserves processes for $\overline{W}$. Furthermore, P'' and Q are disjoint. Since only processes in P take steps in α_1, there are at least $|\overline{W}|$ processes not in P covering each register in $\overline{W} \cap V$ in configuration C'. Hence

$$\begin{aligned}
|P''| &\geq |P| - |\overline{W}| \cdot |\overline{W} \cap (V' - V)| \\
&\geq |\overline{W}| \cdot |\overline{W} \cap \overline{V}| + (r^2 - r + |V| - |V|^2)/2 - |\overline{W}| \cdot |\overline{W} \cap (V' - V)| \\
&= |\overline{W}| \cdot |\overline{W} \cap \overline{V'}| + (r^2 - r + |V| - |V|^2)/2 \\
&\geq |\overline{W}| \cdot |\overline{W} \cap \overline{V'}| + (r^2 - r + |V'| - |V'|^2)/2.
\end{aligned}$$

By the induction hypothesis applied to (V', W), there is an execution γ starting from C' that decides both 0 and 1. Then the execution $\alpha_1\gamma$ starting from C decides both 0 and 1.

Case 2. $V \not\subseteq W$ and $W \not\subseteq V$. Let $U = V \cup W$. Then $V, W \subsetneq U$, so $|\overline{V}|, |\overline{W}| \geq |\overline{U}| + 1$.

Consider configuration C. Since α is interruptible for P and V and reserves processes for $\overline{W}$, there are at least $|\overline{V}| + 1$ processes in P covering each register in V and at least $|\overline{W}|$ processes not in P covering each register in $\overline{W} \cap V$. Let Q'' be a set consisting of $|\overline{W}|$ processes not in P covering each register in $\overline{W} \cap V = U - W$. Similarly, since β is interruptible for Q and W and reserves processes for $\overline{V}$, there are at least $|\overline{W}| + 1$ processes in Q covering each register in W and a set P'' consisting of $|\overline{V}|$ processes not in Q covering each register in $\overline{V} \cap W = U - V$. The processes in P'' and Q'' cover disjoint sets of registers, so $P'' \cap Q'' = \phi$. Let $P' = P \cup P''$ and $Q' = Q \cup Q''$. Since $P \cap Q = \phi$, $P \cap Q'' = \phi$, and $P'' \cap Q = \phi$, it follows that P' and Q' are disjoint.

There are at least $|\overline{V}| \geq |\overline{U}|+1$ processes in P' covering each register in $V \cup (U - V) = U$ and there are at least $|\overline{W}|$ processes not in P' covering each register in $\overline{W} \cap V = \overline{W} \cap U$. Since $|P'| \geq |P| \geq |\overline{W}| \cdot |\overline{W} \cap \overline{V}| + (r^2 - r + |V| - |V|^2)/2 > |\overline{W}| \cdot |\overline{W} \cap \overline{U}| + (r^2 - r + |U| - |U|^2)/2$, Lemma 7.7 implies that there is an interruptible execution α' for P' and

U starting from C that reserves processes for $\overline{W}$. Similarly, $|Q'| > |\overline{V}| \cdot |\overline{V} \cap \overline{U}| + (r^2 - r + |U| - |U|^2)/2$ and there is an interruptible execution β' for Q' and U starting from C that reserves processes for $\overline{V}$.

First suppose that α' decides 0. Since $\overline{U} \subseteq \overline{V}$, execution β is interruptible for Q and W and reserves processes for $\overline{U}$. Then, by the induction hypothesis, there is an execution starting from C that decides both 0 and 1.

Similarly, if β' decides 1, there is an execution starting from C that decides both 0 and 1.

Otherwise, α' decides 1 and β' decides 0. Since $\overline{U} \subseteq \overline{V}, \overline{W}$, it follows that α' is an interruptible execution for P' and U that reserves processes for $\overline{U}$, β' is an interruptible execution for Q' and U that reserves processes for $\overline{U}$, and $|P'|, |Q'| > |\overline{U}| \cdot |\overline{U} \cap \overline{U}| + (r^2 - r + |U| - |U|^2)/2$. Hence, by the induction hypothesis, there is an execution starting from C that decides both 0 and 1. □

Now we prove the main result, by showing that, if there are too many processes compared to the number of registers, then there is a execution in which both 0 and 1 are decided.

Theorem 7.9. *Any consensus algorithm for n processes that uses only registers and satisfies nondeterministic solo termination needs $\Omega(\sqrt{n})$ registers.*

Proof. Suppose there is a consensus algorithm for $n \geq 3r^2 - r + 2$ processes using only r registers. Divide the processes into two sets, P and Q, with more than $(3r^2 - r)/2$ processes each. Let C_0 be an initial configuration in which each process in P has input 0 and each process in Q has input 1. Let $V = W = \phi$. Then $\overline{W} \cap V = \overline{V} \cap W = \phi$ and $|\overline{W}| \cdot |\overline{W} \cap \overline{V}| + (r^2 - r + |V| - |V|^2)/2 = |\overline{V}| \cdot |\overline{V} \cap \overline{W}| + (r^2 - r + |W| - |W|^2)/2 = (3r^2 - r)/2$. Lemma 7.7 implies that, starting from C_0, there is an interruptible execution α for P and V that reserves processes for $\overline{W}$ and an interruptible execution β for Q and W that reserves processes for $\overline{V}$. Since all processes in P have input 0, α must decide 0. Similarly, β must decide 1. Then, by Lemma 7.8, there is an execution starting from C_0 that decides both 0 and 1. This contradicts the correctness of the algorithm. □

These lower bounds, the use of clones, and the concept of nondeterministic solo termination first appeared in the paper *On the Space Complexity of Randomized Synchronization*, by Faith Ellen Fich, Maurice Herlihy, and Nir Shavit, Journal of the ACM, volume 45, 1998, pages 843–862. They also extended these lower bounds to solving consensus using historyless objects.