This lemma, in turn, can be used (within an induction) to prove that for every k, $0 \leq k \leq \log N - 1$, there exists an execution of 2^{k+1} processes with k rounds and ending with a bivalent configuration. Taking $k = \log N - 1$ we get the desired lower bound:

Theorem 13. *In the totally anonymous model, the round complexity of a protocol solving binary consensus among N processes is $\Omega(\log N)$.*

References

1. K. Abrahamson. On achieving consensus using a shared memory. In *Proceedings of the 7th Annual ACM Symposium on Principles of Distributed Computing*, pages 291–302. ACM, 1988.
2. Y. Afek, H. Attiya, D. Dolev, E. Gafni, M. Merritt, and N. Shavit. Atomic snapshots of shared memory. *J. ACM*, 40(4):873–890, Sept. 1993.
3. D. Angluin. Local and global properties in networks of processors. In *Proceedings of the 12th ACM Symposium on Theory of Computing*, pages 82–93, 1980.
4. H. Attiya, M. Snir, and M. Warmuth. Computing on an anonymous ring. *J. ACM*, 35(4):845–876, Oct. 1988.
5. H. Brit and S. Moran. Wait-freedom vs. bounded wait-freedom in public data structures. *Universal Journal of Computer Science*, pages 2–19, Jan. 1996.
6. J. E. Burns and N. A. Lynch. Bounds on shared memory for mutual exclusion. *Information and Computation*, 107(2):171–184, Dec. 1993.
7. D. Dolev, C. Dwork, and L. Stockmeyer. On the minimal synchronism needed for distributed consensus. *J. ACM*, 34(1):77–97, Jan. 1987.
8. M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty processor. *J. ACM*, 32(2):374–382, Apr. 1985.
9. M. J. Fischer, S. Moran, S. Rudich, and G. Taubenfeld. The wakeup problem. *SIAM J. Comput.*, 25(6):1332–1357, Dec. 1996.
10. M. P. Herlihy. Wait-free synchronization. *ACM Trans. Prog. Lang. Syst.*, 13(1):124–149, Jan. 1991.
11. P. Jayanti and S. Toueg. Wakeup under read/write atomicity. In J. van Leeuwen and N. Santoro, editors, *Proceedings of the 4th International Workshop on Distributed Algorithms*, volume 486 of *Lecture Notes in Computer Science*, pages 277–288. Springer-Verlag, 1990.
12. M. C. Loui and H. H. Abu-Amara. Memory requirements for agreement among unreliable asynchronous processes. In *Advances in Computing Research, Vol. 4*, pages 163–183. JAI Press, Inc., 1987.
13. S. Moran and G. Taubenfeld. A lower bound on wait-free counting. *Journal of Algorithms*, 24:1–19, 1997.
14. S. Moran and Y. Wolfsthal. An extended impossibility result for asynchronous complete networks. *Inf. Process. Lett.*, 26:141–151, 1987.
15. M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, Apr. 1980.

This article was processed using the LaTeX macro package with LLNCS style

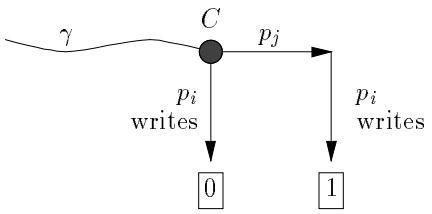

Fig. 2. p_i writes to r.

Similarly, $(C, p_i) \preceq (C', p_j \cdot p_i)$. Since $p_i(C)$ is 0-valent, for every p_j-free schedule σ'', the decision in $(\pi(C'), \sigma'')$ is 0.

Therefore, $\pi(C')$ is bivalent (see Figure 3).

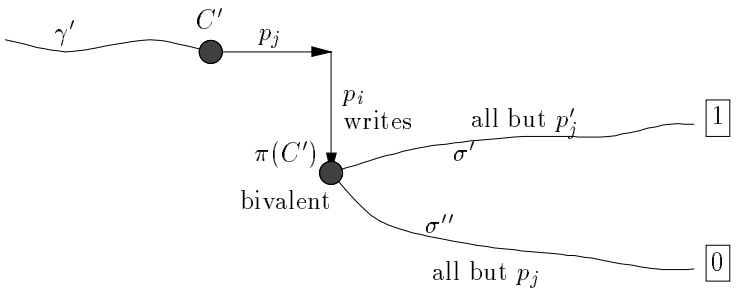

Fig. 3. The contradiction when p_i writes to r.

Let β be the execution extending γ' with the schedule π; $n+1$ processes participate in β; β ends with a bivalent configuration. Since γ includes k rounds, γ' also includes k complete rounds (by Lemma 9). Therefore, $p_i, p_j \notin NY_{k+1}(\beta)$. For every process $p \neq p_i, p_j, p'_j$, by Lemma 9, if $p \in NY_{k+1}(\gamma')$ then $p \in NY_{k+1}(\gamma)$ and thus $p \in NY_{k+1}(\alpha)$.

If $p_j \notin NY_{k+1}(\alpha)$ then $p'_j \notin NY_{k+1}(\beta)$ and thus, since $p_i \in NY_{k+1}(\alpha)$, $|NY_{k+1}(\beta)| < |NY_{k+1}(\alpha)|$. If $p_j \in NY_{k+1}(\alpha)$, then it is possible that $p'_j \in NY_{k+1}(\beta)$, but since $p_i \in NY_{k+1}(\alpha)$, $|NY_{k+1}(\beta)| < |NY_{k+1}(\alpha)|$.

The other case, when p_i reads from r (and p_j writes to r), follows by similar arguments, multiplying p_i and taking $\pi = p_i \cdot p_j \cdot p'_i$. □

Lemma 11 can be used (within an induction) to prove the next lemma:

Lemma 12. *Let α be an execution of $n \leq N/2$ processes with k rounds that ends with a bivalent configuration. Then there exists an execution β of $\leq 2n$ processes with $k+1$ rounds that ends with a bivalent configuration.*

Lemma 9. *Let σ be a schedule of $q_1, q_2, \ldots, q_n$ with k rounds and let τ be the multiplication of σ by $\{P_1, \ldots, P_n\}$. Then τ has k rounds (of $\cup_{i=1}^{n} P_i$). Moreover, if some process $p \in P_i$ is in $NY_{k+1}(\tau)$ then q_i is in $NY_{k+1}(\sigma)$.*

A configuration C is *critical* if there are two processes p and q such that a step of p in C leads to a v-valent configuration and the pair of steps q and then p leads to a $\bar{v}$-valent configuration. We say that p is a *critical leader* in C. The following lemma is from [8].

Lemma 10. *Let C be a bivalent configuration and let p be a process. Then there is a finite schedule σ such that either a step of p from $\sigma(C)$ leads to a bivalent configuration or p is a critical leader in $\sigma(C)$.*

Given a protocol solving consensus, we construct, in iterations, an execution that includes "many" rounds. In each iteration, the execution contains one more round; all executions end with a bivalent configuration.

Lemma 11. *Let α be an execution of $n < N$ processes with k rounds that ends with a bivalent configuration. Then there exists an execution β of at most $n+1$ processes with k rounds that ends with a bivalent configuration, such that $|NY_{k+1}(\beta)| < |NY_{k+1}(\alpha)|$.*

Proof. Let D be the final configuration in α. Since D is bivalent and α includes exactly k rounds, $NY_{k+1}(\alpha)$ is not empty. Let p_i be a process from $NY_{k+1}(\alpha)$. By Lemma 10, there is a finite schedule τ such that either a step of p_i in $\tau(D)$ leads to a bivalent configuration or $\tau(D)$ is a critical configuration and p_i is a critical leader in $\tau(D)$.

Let γ be the execution that extends α with schedule τ and assume it ends with the configuration C.

If a step of p_i from C leads to a bivalent configuration, then let β be the extension of γ with a step of p_i. All requirements hold: n processes participate in β; β ends with a bivalent configuration, by assumption; β is an extension of α and therefore includes k complete rounds; $NY_{k+1}(\beta)$ includes at least one process less than $NY_{k+1}(\alpha)$, that is, p_i. Therefore, β is the required execution.

Otherwise, p_i is a critical leader in $C = C_{end}(\gamma)$; let p_j be the other process that takes part in the "critical" extension.

Consider the steps of p_i and p_j from C. Simple arguments, which are omitted, show that $i \neq j$, that p_i and p_j access the same register r and that at least one of them writes to r.

If p_i writes to r, then consider an execution γ', ending with a configuration C', that is the multiplication of γ by $\{\{p_1\}, \ldots, \{p_i\}, \ldots, \{p_j, p_j'\}, \ldots, \{p_n\}\}$. By Corollary 2, $\gamma' \approx \gamma$.

Consider the schedule $\pi = p_j \cdot p_i$ (see Figure 2).

Since $(\gamma', p_i) \sim (\gamma, p_i)$, $(\gamma', p_j) \sim (\gamma, p_j)$, $(\gamma', p_j') \sim (\gamma, p_j)$, the step of p_i is a write to r and p_j also accesses r, $(C, p_j \cdot p_i) \preceq (C', p_j \cdot p_i)$. Since $p_j \cdot p_i(C)$ is 1-valent, for every p_j'-free schedule σ', the decision in $(\pi(C'), \sigma')$ is 1.

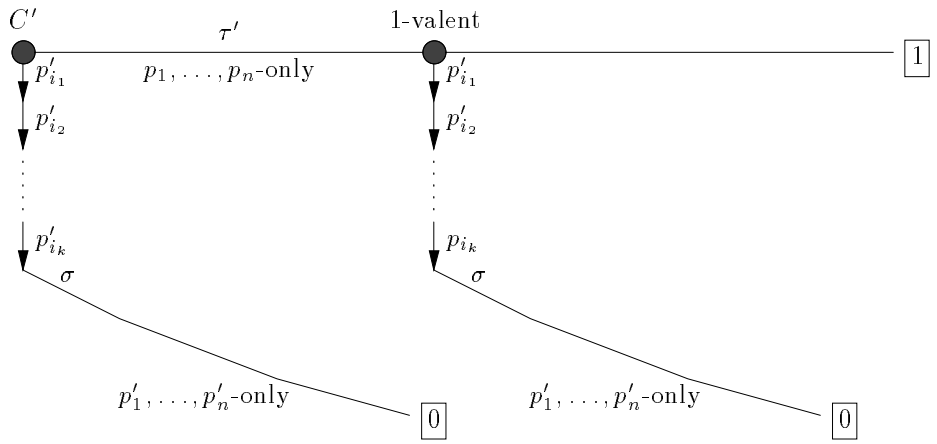

Fig. 1. Illustration for the proof of Lemma 7.

Theorem 8. *In the totally anonymous model, a protocol solving consensus among N processes requires $\Omega(\log N)$ shared registers.*

5.2 Lower Bound on the Round Complexity

Given a schedule σ of $p_1, \ldots, p_n$, a *round* in σ is a minimal segment (consecutive subsequence) of σ in which each process appears at least once. The *number of rounds* in a schedule σ is the maximal k such that σ includes k disjoint rounds. If σ includes k rounds, then it can be written as a concatenation $\sigma_1, \ldots, \sigma_k \sigma'$, where the (possibly empty) schedule σ' is not a round. If σ' is not empty, then we refer to it as the $(k+1)$-st *incomplete* round of σ; otherwise, we say that the final (k-th) round of σ is *complete*.

The *round complexity of an execution* α is the number of rounds in the schedule corresponding to the maximal finite prefix of α in which no process decides; the rounds of an execution are induced in a natural way from the rounds of the corresponding schedule. The *round complexity of a protocol* is the maximal round complexity, over all the executions of the protocol. It can be verified that the round complexity of the binary consensus protocol presented in Section 4.2 is $O(N)$.

Let $\mathcal{P}$ be a binary consensus protocol for N processes.

For a schedule σ of $p_1, \ldots, p_n$ with k rounds, $NY_{k+1}(\sigma)$ ("not yet" in round $k+1$ of σ) denotes the set of processes that do not appear in the incomplete round of σ; if the last round of σ is complete, then $NY_{k+1}(\sigma) = \{p_1, \ldots, p_n\}$. The next simple lemma, whose proof is omitted, shows that multiplication (defined in Section 3) preserves the number of rounds and the sets NY.

waiting processes cover all registers written in all previous iterations. Thus, if no new register is written, waiting processes may overwrite all registers and continue as if the current iteration never happened. That is, a "new" register must be written, and the execution can be extended with one more covered register.

The next lemma is the inductive step of the lower bound proof.

Lemma 7. *Let α be a finite execution of $n \leq N/2$ processes, that ends with a bivalent configuration in which k registers are covered, and all writes in α are to covered registers. Then there exists a finite execution β of $2n$ processes, that ends with a bivalent configuration in which $k+1$ registers are covered, and all writes in β are to covered registers.*

Proof. Let α be a finite execution of $q_1, \ldots, q_n$ satisfying the conditions of the lemma. Let α' be the multiplication of α by $\{\{p_1, p'_1\}, \ldots, \{p_n, p'_n\}\}$. Assume α ends in configuration C and α' ends in configuration C'.

By Corollary 2, $\alpha \approx \alpha'$, and thus all writes in α' are to the k covered registers, denoted $r_1, \ldots, r_k$. By Lemma 1, $(\alpha, q_i) \sim (\alpha', p'_i)$ and thus there are k processes $p'_{i_1}, \ldots, p'_{i_k}$ that cover $r_1, \ldots, r_k$ in C'.

Consider a $\{p'_1, \ldots, p'_n\}$-only schedule σ starting with the sequence $p'_{i_1} \cdot p'_{i_2} \cdots p'_{i_k}$ (each of the covering processes takes one step). Since in every $\{q_1, \ldots, q_n\}$-only fair execution all processes eventually decide, in every $\{p'_1, \ldots, p'_n\}$-only fair execution all processes eventually decide. Without loss of generality, assume that the decision in (C', σ) is 0.

Since C is bivalent, there is a $\{q_1, \ldots, q_n\}$-only schedule ρ such that the decision in (C, ρ) is 1. Since $(\alpha, q_i) \sim (\alpha', p_i)$ for all i, there is a $\{p_1, \ldots, p_n\}$-only schedule τ such that the decision in (C', τ) is 1.

Assume there is a prefix τ' of τ such that $\tau'(C')$ is univalent and the only writes in (C', τ') are to $r_1, \ldots, r_k$. Since the decision in (C', τ) is 1, $\tau'(C')$ must be 1-valent. Consider the execution $(\tau'(C'), \sigma)$. Since all writes in (C', τ') are to $r_1, \ldots, r_k$, after the k writes to the registers $r_1, \ldots, r_k$, the memory state is the same as in the execution from C' and the states of $p'_1, \ldots, p'_n$ are the same. Therefore, the decision in $(\tau'(C'), \sigma)$ is also 0, contradicting fact that $\tau'(C')$ is 1-valent. (See Figure 1.)

Therefore, there must be a write to a register other than $r_1, \ldots, r_k$ in (C', τ) and the configuration before the first such write is bivalent. Let τ'' be the longest prefix of τ that do not include the first write to $r_{k+1} \notin \{r_1, \ldots, r_k\}$. Let β be the execution extending α' with the schedule τ''.

In β, $2n$ processes participate; β ends with a bivalent configuration; $k+1$ registers are covered after β: $r_1, \ldots, r_k$ are covered by $p'_{i_1}, \ldots, p'_{i_k}$ and r_{k+1} is covered by the construction; all writes in β are to $r_1, \ldots, r_{k+1}$, by construction. Thus, β is the required execution. □

By careful induction on k, in which Lemma 6 provides the base case, and Lemma 7 provides the inductive step, we can prove that for every k, $0 \leq k < \log N$, there exists an execution α_k of 2^{k+1} processes ending with a bivalent configuration, such that k registers are covered after α_k. Taking $k = \log N - 1$, we get:

with input x do not participate in later iterations; if the decisions after j iterations are $set(\overline{Z})$, for some valid input vector $\overline{Z}$, then a process that participates in the j-th iteration writes the set to the decision register; a process that stops participating at some stage, reads the decision register until it finds a non-empty value.

The detailed code and correctness proof for the protocol are omitted.

To prove that the condition is necessary, consider a relation R, with an input vector $\overline{X}$ such that $\bigcap\{R[\overline{Y}] \mid \overline{Y} \in \mathcal{D}_R \text{ and } set(\overline{Y}) \supseteq set(\overline{X})\} \neq \Phi$, and let $set(\overline{X})$ be $\{x_1, x_2, \ldots, x_k\}$.

Suppose for contradiction that there is a protocol computing R.

We let k processes with inputs $x_1, \ldots, x_k$ run on their own; by Lemma 3, they eventually decide on $z \in R[\overline{X}]$.

By the assumption on $\overline{X}$, there exists $\overline{Y} \in \mathcal{D}_R$ such that $set(\overline{Y}) \supseteq set(\overline{X})$ and $z \notin R[\overline{Y}]$; otherwise, z is in the intersection, which violates the assumption on $\overline{X}$. Without loss of generality, $\overline{Y} = (x_1, x_2, \ldots, x_k, x_{k+1}, \ldots, x_r)$.

Now, let processes with inputs $x_{k+1}, \ldots, x_r$ run, extending the execution of the first k processes. They have to decide on some value from $R[\overline{Y}]$, but the first k processes decide on $z \notin R[\overline{Y}]$, which is a contradiction. The details appear in the full version.

5 Lower Bounds for Consensus

5.1 Lower Bound on the Number of Registers

Following Fischer, Lynch and Paterson [8], a configuration C is *0-valent* if only 0 is decided in executions from C; similarly, C is *1-valent* if only 1 is decided in executions from C. C is *univalent* if it is either 0-valent or 1-valent, and is *bivalent* otherwise. Although processes do not fail, we can prove the following lemma:

Lemma 6. *There is an initial configuration with two processes which is bivalent.*

Proof. Consider an initial configuration, C_0, with two processes—p_0 with an input 0 and p_1 with an input 1. $\overline{X} = (0, \ldots, 0)$ is an N-entry input vector for consensus; by Lemma 3, p_0 decides 0 in an infinite p_0-only execution from C_0. Similarly, $\overline{Y} = (1, \ldots, 1)$ is an N-entry input vector for consensus; by Lemma 3, p_1 decides on 1 in an infinite p_1-only execution from C_0. Therefore, C_0 is bivalent. □

Following Burns and Lynch [6], we say that a process p *covers* a register r in a configuration C if the step of p from C is a write to r. In order to prove the lower bound on the number of registers, we construct an execution in which "many" registers are covered. The construction is iterative; in each iteration, one more register is covered.

In each iteration a process can be *active*, i.e., take steps in the current iteration, or *waiting*, i.e., take no steps in the current iteration. At each iteration,

ReduceOne with the output from the current copy, and follows the same rules. When a process gets a non-$\perp$ output in the $(N-1)$-st copy of ReduceOne, it writes this value to the decision register and decides on it; a process waiting for a non-empty value in the decision register, decides upon reading this value.

It is guaranteed that at least one process participates in every copy of ReduceOne; in particular, at least one process performs the $(N-1)$-st copy; moreover, the inputs to all copies of ReduceOne are valid, and all processes have the same output in the $N-1$-st copy of ReduceOne.

If N is not known, then we modify the protocol so that each process checks, before starting the i-th copy, whether a process with an opposite value already participated in this copy; if so, it waits to read a non-empty value from the decision register. Moreover, a process proceeds to the $(i+1)$-st copy only if some other process performed the $(i-1)$-st copy with an opposite input; otherwise, the processor writes its current output to the decision register and terminates with this value. The detailed code and precise correctness proof for the protocol, which are based on the above ideas, are omitted.

There are N copies of ReduceOne, each requiring two shared registers; thus, the protocol requires $O(N)$ shared registers.

Non-binary consensus is achieved by agreeing (using binary consensus) on each bit of the output. Once again, the details are omitted.

4.3 Relations

Finally, we characterize which relations can be computed in our model.

Theorem 5. *A relation R can be computed in the totally anonymous model if and only if for every input vector $\overline{X}$ it holds that $\bigcap\{R[\overline{Y}] \mid \overline{Y} \in \mathcal{D}_R$ and $set(\overline{Y}) \supseteq set(\overline{X})\} \neq \Phi$.*

The consensus problem satisfies this condition and, indeed, it can be computed. The characterization for functions (Theorem 4) is a special case of Theorem 5.

To prove that the condition is sufficient, consider a relation R satisfying the assumption of Theorem 5.

The *tower* of $set(\overline{X})$ is $\{set(\overline{Y} \mid \overline{Y} \in \mathcal{D}_R$ and $set(\overline{Y} \supseteq set(\overline{X})\}$. Every input set is in some tower, and the union of the towers is the input domain of the relation. Note that the towers do not depend on the output values.

By the assumption on R, each tower has an output value that is legal for all inputs in the tower. Thus, if processes agree on a tower that the input vector belongs to, they can compute R; it suffices to agree on some set in the tower.

The idea is to reach consensus on some subset of the input set. If the subset includes the input set of some legal input vector, this determines the output value, otherwise there are values that can be added to the subset.

The protocol works in iterations; in each iteration, the protocol of the previous section is used to solve consensus. Each process starts the first iteration with its input; if the decision in the j-th consensus protocol is x, then processes

registers (each time more and more registers), until it obtains a determining set Z, and outputs $val(Z)$.

It is not guaranteed that all processes see the same determining set, but if a process sees a determining set, then processes obtaining a later determining set will have the same output value. Moreover, this value must be $f(\overline{X})$, where $\overline{X}$ is the input vector. Processes halt when all input values have been written (at the latest), since $set(\overline{X})$ is a determining set.

The protocol can be modified to use only two registers. One register is used to collect all the input values known so far, while the other is used to announce a decision value. As long as no decision value is announced, a process makes sure that the other register contains all input values it knows of. The detailed code and correctness proof of this protocol are omitted.

In the full version, we also prove that there are functions which cannot be computed with a single register.

To prove that the condition is necessary, consider some function f, for which there is a pair of input vector $\overline{X}, \overline{Y}$ such that $set(\overline{X}) \subseteq set(\overline{Y})$ and $f[\overline{X}] \neq f[\overline{Y}]$. Let $set(\overline{X}) = \{x_1, \ldots, x_k\}$ and let $\overline{Y} = (x_1, \ldots, x_k, x_{k+1}, \ldots, x_r)$.

Suppose for contradiction that there is a protocol for computing f.

Informally, we let k processes with inputs $x_1, \ldots, x_k$ run on their own; by Lemma 3, eventually they decide on $f[\overline{X}]$. Then we let processes with inputs $x_{k+1}, \ldots, x_r$ run on their own; they have to decide on $f[\overline{Y}] \neq f[\overline{X}]$. Therefore, the decision of the first k processes is incorrect. The formal arguments are left to the full version.

4.2 Consensus

We outline how consensus can be solved in our model, starting with the binary case.

The protocol for binary consensus employs a procedure ReduceOne that reduces the "disagreement" in processes' inputs. The procedure uses two binary shared registers, $awake[0]$ and $awake[1]$, both initially $false$. A process with input x writes $true$ to $awake[x]$ and reads the other register. If the other register is $false$, the procedure returns x; if the other register is $true$ and $x = 0$, the procedure returns $\perp$; if the other register is $true$ and $x = 1$, then the procedure returns 0. ReduceOne guarantees the following properties:

- The output of at least one process is not $\perp$.
- If the output of p_i is $v \neq \perp$, then there exists some process whose input is v.
- If processes output both 0 and 1, then the number of processes with output 1 is strictly less than the number of processes with input 1.

When the number of processes, N, is known, ReduceOne can be used to solve consensus in the following simple way.

Take $N-1$ copies of procedure ReduceOne. Each process starts the first copy of ReduceOne with its input; if it gets $\perp$, then it waits to read a non-empty value from a separate decision register; otherwise, it starts the next copy of

Corollary 2. *Let α be an execution of $q_1, \ldots, q_n$. If β is a multiplication of α, then $\alpha \approx \beta$.*

Note that if $\alpha \approx \beta$, then the same set of shared registers is accessed in α and in β.

Since the model is anonymous, neither the order of values in the input vector nor the multiplicity of a value are significant. The following definition and lemma formalize this property.

If $\overline{X} = (x_1, x_2, \ldots, x_N)$ is an input vector, then let $set(\overline{X})$ be the set of distinct values in $\overline{X}$. The next lemma shows that for any input vector $\overline{X}$, if we run processes with a single appearance of each input value then processes have to decide on a value in $R[\overline{X}]$.

Lemma 3. *Let R be a relation, let $\mathcal{P}$ be a protocol that computes R, and let $\overline{X}$ be an N-entry input vector of R with n distinct values. If $\overline{Z}$ is an n-entry vector such that $set(\overline{Z}) = set(\overline{X})$, then in every fair execution of n processes with input vector $\overline{Z}$ all processes decide on $y \in R[\overline{X}]$.*

4 Computability Results

4.1 Functions

We start with a characterization of the functions that can be computed in our model. A relation R is a *function* if a single output value is allowed for every legal input vector.

Theorem 4. *A function f can be computed in the totally anonymous model if and only if for every pair of input vectors $\overline{X}, \overline{Y} \in \mathcal{D}_f$ such that $set(\overline{X}) \subseteq set(\overline{Y})$ it holds that $f[\overline{X}] = f[\overline{Y}]$.*

An example is computing the minimum of N input values where exactly k values are different (for some known constant k). Note that the multiplicity of values in the input vector does not affect the function's value, and thus the characterization only refers to the input sets.

To prove that the condition is sufficient, consider a function f satisfying the assumption of Theorem 4.

A set $Z = \{z_1, \ldots, z_k\}$ is *determining* for f if there exists an output value $val(Z)$ such that for every input vector $\overline{X} \in \mathcal{D}_f$, if $Z \subseteq set(\overline{X})$ then $f[\overline{X}] = val(Z)$.

Intuitively, the idea is to find a determining set that allows to compute the function. A simple protocol uses an unbounded number of Boolean registers, one for each possible input value,[4] initialized to *false*. A process announces its input by setting the appropriate register to *true*, and then repeatedly reads the

[4] The protocol assumes that the set of possible input values is enumerable and, for simplicity, refers to them as integers.

Given a schedule σ, a configuration C and a protocol $\mathcal{P}$, the execution segment that results from applying steps of processes according to σ to the configuration C is denoted (C, σ).

An infinite schedule σ is *fair* if every process occurs infinitely many times in σ. An infinite execution α is *fair* if the corresponding schedule is fair.

A *task* is a set of pairs, each containing an input vector and a set of output vectors that are allowable for this input vector.

We restrict our attention to *relations* which are tasks in which all processes have to agree on the same output value; in this case, all entries of an output vector are equal. A vector is a *legal input vector* for a relation R if it appears in the left-hand side of some pair in R; the *domain* of R, denoted $\mathcal{D}_R$, contains the legal input vectors for R.

A typical relation is *k-valued consensus*; the inputs are vectors of integers from $\{0, \ldots, k-1\}$, and the allowable output values for an input vector $\overline{X}$ are all the values in $\overline{X}$.

A protocol $\mathcal{P}$ *computes* a relation R if for every legal input vector $\overline{X} = (x_1, x_2, \ldots, x_N)$ and for every fair execution α of $\mathcal{P}$ in which process p_i has an input x_i, every process eventually writes the same *decision value* $d \in R[\overline{X}]$ into its local output register.

3 Basic Properties

A basic property of our model is that a single process can be replaced by a group of processes where each process has the same input, and vice versa. In this section, we define notions of *multiplication* and *equivalence*, capturing this property. The impossibility results and lower bounds presented later rely on these notions.

Given a finite execution α, let the *view* of process p after α, denoted $\alpha|p$, be the vector containing the internal state of p and the values of the shared registers in $C_{end}(\alpha)$. If α and β are finite executions and p and q are processes appearing in α and β, respectively, then we write $(\alpha, p) \sim (\beta, q)$, if $\alpha|p = \beta|q$.

A finite execution α of $p_1, \ldots, p_k$ *hides* a finite execution β of $q_1, \ldots, q_n$, denoted $\alpha \preceq \beta$, if for every p_i there exists q_j such that $(\alpha, p_i) \sim (\beta, q_j)$. If $\alpha \preceq \beta$ and $\beta \preceq \alpha$, i.e., they hide each other, then α and β are *equivalent*, denoted $\alpha \approx \beta$. This notion extends to infinite executions if it holds after infinitely many finite prefixes of them.

Let σ be a schedule of $q_1, \ldots, q_n$, and let $\{P_1, \ldots, P_n\}$ be a set of groups of processes, where $P_i = \{p_i^1, p_i^2, \ldots, p_i^{l(i)}\}$, for every i, $1 \leq i \leq n$. The *multiplication* of σ by $\{P_1, \ldots, P_n\}$ is the schedule obtained from σ by substituting every appearance of q_i in σ with the sequence $p_i^1, p_i^2, \ldots, p_i^{l(i)}$. Let α be an execution whose schedule is σ, such that q_i has an input x_i in α. The *multiplication* of α by $\{P_1, \ldots, P_n\}$ is the execution obtained by letting each process in P_i start with input x_i and applying the multiplication of σ.

Lemma 1. *Let α be an execution of $q_1, \ldots, q_n$. If β is the multiplication of α by $\{P_1, \ldots, P_n\}$, then $(\alpha, q_i) \sim (\beta, p)$, for every i, $1 \leq i \leq n$, and for every $p \in P_i$.*

The second model is similar to ours, but the shared memory is not initialized to a known state. In the first model, the *wakeup* problem [9] and the consensus problem can be solved, but the leader election problem cannot be solved; in the second model, all these problems, and in particular, consensus, cannot be solved.

2 Outline of the Model

We consider a standard asynchronous system of $N \geq 2$ anonymous non-faulty processes which communicate via read/write shared registers. For ease of exposition, we refer to the processes by the unique names $p_1, p_2, \ldots, p_N$, but the names are not available to the processes. Every process can read from or write to any shared register.

Each process has an *internal state*, including *input* and *output* registers and an unbounded amount of internal storage. Each process acts according to a *transition function*, determining for each internal state, the new internal state and the next action—whether this is a read or a write, to which register and what is the value written.

An *initial state* prescribes starting values to the input register and the internal storage, and the *empty* value to the output register. A process is in a *terminated state* if its output register is not empty.

A *protocol* specifies for each process the initial value of the internal storage and a transition function; we assume these are the same for all processes. We assume that the protocol is *uniform* and the total number of processes in the system is unknown, unless stated otherwise.

A *configuration* consists of the internal states of all processes, together with the values of the shared registers. An *initial configuration* of a protocol $\mathcal{P}$ requires every process to be in the initial state prescribed by $\mathcal{P}$ and the shared registers to contain default values.

The computation proceeds by *steps*, each by a single process, taking one configuration to a successive configuration; a step is a read or a write from a single register, or a *nop*. Read and write steps can be taken only if the process is not in a terminated state; nop steps can be taken only if the process is in a terminated state. If p is a process and C is a configuration, then $p(C)$ denotes the configuration obtained after p takes a step in C according to the transition function of a specific protocol $\mathcal{P}$.

A *schedule* is a sequence of process names. If $\sigma = p_{i_1}, p_{i_2}, \ldots, p_{i_n}$ is a finite schedule and C is a configuration, then $\sigma(C)$ denotes $p_{i_n}(p_{i_{n-1}}(\ldots p_{i_1}(C)\ldots))$.

An *execution* of a protocol $\mathcal{P}$ is a (finite or infinite) sequence of the form

$$C_0, p_{i_1}, C_1, p_{i_2}, C_2, p_{i_3}, \ldots,$$

where C_0 is an initial configuration and for every $k \geq 1$, C_k is a configuration, p_{i_k} is a process, and C_k is $p_{i_k}(C_{k-1})$. If α is a finite execution, then $C_{end}(\alpha)$ is the last configuration in α.

media cannot distinguish between different processes. This models an environment where the communication route connecting a process to the shared server can be dynamically replaced by an alternative one.

We will be mainly interested in the computation power of this model, namely, what kind of distributed tasks [14] can be computed by it. We restrict our attention to consensus-like tasks, in which each process has a private input, and processes have to agree on the same output. Such a task defines a *relation* between input vectors and possible output values. This relation is a *function* if the corresponding output is uniquely determined by the processes inputs (e.g., the AND function); note that the standard consensus task [15] is not a function.

First we give a complete characterization of the functions that can be computed within this model, and show that if a function can be computed, then it can be computed with only two shared registers. It turns out that the class of functions that can be computed is quite limited (for instance, the AND or OR functions cannot be computed); in view of this, it is a little surprising that consensus is computable in this model. We will provide a (fault-free) consensus protocol in which both the number of shared registers and the number of rounds is linear in the number of processes. Using this protocol, we then give a complete characterization of the relations that can be computed.

One can ask whether consensus can be solved in the totally anonymous model with a smaller number of shared registers, and/or smaller number of rounds. We give a partial answer to this question by showing that these figures cannot be bounded by a constant independent of the number of processes. Specifically, we prove logarithmic lower bounds on the number of shared registers and the number of rounds needed for solving consensus.

Our lower bound proofs combine two seemingly unrelated techniques:

The first one is of "covering" registers, which is similar to the technique used by Burns and Lynch [6] to prove that any protocol for mutual exclusion among N processes needs N shared registers. Different variants of this technique appear in the "hiding lemma" used by Moran and Taubenfeld [13] to prove impossibility of wait-free counting, and in the "reconstruction lemma" used in the context of public data structures by Brit and Moran [5].

The other tool we use in our lower bound proofs is the existence of bivalent states, introduced by Fischer, Lynch and Paterson [8] to prove the impossibility of asynchronous consensus in the presence of one faulty process. This tool was used extensively afterwards, e.g., [7, 10, 12], mainly in the context of impossibility results in fault tolerant computations. Thus, our results demonstrate a certain connection between faulty models with distinct processes, and non-faulty models with totally anonymous processes.

Many works on shared memory systems studied models with faults [1, 2, 10, 12]. These papers assume that processes are not anonymous, and may run different protocols.

Jayanti and Toueg [11] studied anonymous read/write shared memory models. They considered two models: In the first model, the processes have no names but the shared registers are single writer (each process has its "own" register).

Computing in Totally Anonymous Asynchronous Shared Memory Systems
(Extended Abstract)

Hagit Attiya* Alla Gorbach** and Shlomo Moran***

Department of Computer Science, The Technion

Abstract. In the *totally anonymous* shared memory model of asynchronous distributed computing, processes have no id's and run identical programs. Moreover, processes have identical interface to the shared memory, and in particular, there are no *single-writer* registers. This paper assumes that processes do not fail, and the shared memory consists only of read/write registers, which are initialized to some default value. A complete characterization of the functions and relations that can be computed within this model is presented. The consensus problem is an important relation which can be computed. Unlike functions, which can be computed with two registers, the consensus protocol uses a linear number of shared registers and rounds.
The paper proves logarithmic lower bounds on the number of registers and rounds needed for solving consensus in this model, indicating the difficulty of computing relations in this model.

1 Introduction

In this work, we study the *totally anonymous shared memory* model of asynchronous distributed computing. Like in previous works that studied computations by identical processes, e.g., [3, 4], we assume that processes have no id's and run identical programs. We also assume that the means by which processes access the shared memory are identical to all processes. This implies, for example, that a process cannot have a private register to which only this process may write and all other processes can read, as is usually assumed. In this work we study only the case where processes do not fail, and the shared memory consists only of read/write shared registers, which are initialized to some default value. In some cases, we also assume that the number of processes that execute a given task is unknown.

This model is related to an environment where an unknown set of processes execute a common task using a "public" server, in which the communication

* Email: hagit@cs.technion.ac.il.
** Email: allag@msil.sps.mot.com.
*** Email: moran@cs.technion.ac.il. Research supported by the Bernard Elkin Chair for Computer Science. Part of the work was done while this author was at the University of Arizona, supported by US-Israel BSF grant 95-00238.