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

[FM88] P. Feldman and S. Micali. Optimal algorithms for byzantine agreement. In *Proc. 20th ACM Symp. on Theory of Computing*, pages 148–161, 1988.

[GY89] R. L. Graham and A. C. Yao. On the improbability of reaching byzantine agreements. In *Proc. 21st ACM Symp. on Theory of Computing*, pages 467–478, 1989.

[IR81] A. Itai and M. Rodeh. The lord of the ring, or probabilistic methods for breaking symmetry in distributed networks. In *Proc. 22th IEEE Symp. on Foundations of Computer Science*, pages 150–158, Nashville, Tennessee, October 1981.

[KR92] E. Kushilevitz and M. O. Rabin. Randomized mutual exclusion algorithms revisited. In *Proc. 11th ACM Symp. on Principles of Distributed Computing*, pages 275–283, 1992.

[KY84] A. Karlin and A. C. Yao. Probabilistic lower bounds for byzantine agreement. 1984.

[LR81] D. Lehman and M. O. Rabin. On the advantage of free choice: A symmetric and fully distributed solution to the dining philosophers problem. In *Proc. 8th POPL*, pages 133–138, 1981.

[Rab82] M. O. Rabin. n-process mutual exclusion with bounded waiting by $4 \log_2 n$-valued shared variable. *JCSS*, 25(1):66–75, 1982.

[Sai92] I. Saias. Proving probabilistic correctness statements: The case of Rabin's algorithm for mutual exclusion. In *Proc. 11th ACM Symp. on Principles of Distributed Computing*, pages 263–272, 1992.

[Yao77] A. C. Yao. Probabilistic computations: Toward a unified measure of complexity. In *Proc. 18th IEEE Symp. on Foundations of Computer Science*, pages 222–227, 1977.

Proof: Again, we show a lottery with $f(t) = O(t^c)$; by Theorem 13, this completes the proof. Consider the following lottery: the value j is chosen with probability $1/2^{c^j}$, for $(j = 1, 2, ..., c' \log\log n)$, and the value 0 is chosen otherwise. For every integer t (the number of participants), let $\ell \geq 0$ be an integer such that $2^{c^\ell} \leq t < 2^{c^{\ell+1}}$ (the constant c' is chosen so as to guarantee that such an ℓ exists for every $t \leq n$). We are interested in the event that P_i chooses a value $\ell + 1$ and all other $t - 1$ participating processes choose value at most ℓ. This clearly lower bounds the probability that P_i is the unique process that chooses the maximum value. The probability that P_i chooses the value $\ell + 1$ is $1/2^{c^{\ell+1}}$. For each P_j, $j \neq i$, the probability that P_j chooses a value greater or equal to $\ell + 1$ is

$$\sum_{k=\ell+1}^{c' \log\log n} \frac{1}{2^{c^k}} \leq \sum_{k=\ell+1}^{c' \log\log n} \left(\frac{1}{2^c}\right)^k < \frac{c''}{2^{c^{\ell+1}}},$$

where c'' is a constant (e.g., $c'' = 2\lceil 1/\log_2 c \rceil$ suffices). Therefore, the probability that P_j chooses a value less than or equal to ℓ is at least $1 - \frac{c''}{2^{c^{\ell+1}}}$. Since we have $t-1$ different P_j's, the probability that P_i chooses $\ell + 1$ and all other $t - 1$ participating processes choose values of at most ℓ is at least

$$\frac{1}{2^{c^{\ell+1}}} \cdot \left(1 - \frac{c''}{2^{c^{\ell+1}}}\right)^{t-1} \geq \frac{1}{2^{c^{\ell+1}}} \cdot \left(1 - \frac{c''}{2^{c^{\ell+1}}}\right)^{2^{c^{\ell+1}}} \geq \frac{1}{t^c} \cdot e^{-c''},$$

which completes the proof of the theorem. (The proof remains similar in the case that we wish to get a lottery with $f(t) = \alpha t^c$ for a *particular* constant α.) □

References

[AS91] H. Attiya and M. Snir. Better computing on the anonymous ring. *Journal of Algorithms*, 12:204–238, June 1991.

[Ben83] M. Ben-Or. Another advantage of free choice: Complete asynchronous agreement protocols. In *Proc. 6th ACM Symp. on Principles of Distributed Computing*, pages 27–30, August 1983.

[BFJ+82] J. E. Burns, M. J. Fischer, P. Jackson, N. A. Lynch, and G. L. Peterson. Data requirements for implementation of n-process mutual exclusion using a single shared variable. *JACM*, 29:183–205, 1982.

[Bra85] G. Bracha. An $O(\log n)$ expected rounds randomized byzantine generals protocol. In *Proc. 17th ACM Symp. on Theory of Computing*, pages 316–326, 1985.

[CIL87] B. Chor, A. Israeli, and M. Li. On process coordination using asynchronous hardware. In *Proc. 6th ACM Symp. on Principles of Distributed Computing*, pages 86–97, August 1987.

[Dij65] E. Dijkstra. Solution of a problem in concurrent programming control. *CACM*, 8(9):569, 1965.

[FL82] M. Fischer and N. Lynch. A lower bound for the time to assure interactive consistency. *IPL*, 14(4):183–186, 1982.

5 Upper Bounds

In this section, we present some upper bounds to complete the picture. In fact, we do not explicitly present protocols. Instead, we present appropriate *lotteries*, where a lottery is just a probability distribution that allows processes drawing numbers ("tickets") in $\{1, 2, \ldots, B\}$. The "winners" of the lottery are those processes drew the maximal drawn number. We use as a *black-box* the following theorem, implicit in [KR92], that reduce the existence of mutual-exclusion algorithms with certain fairness properties to the existence of lotteries that guarantee a certain probability of having a *unique winner*. More precisely,

Theorem 13: *[KR92] Let f be a function, n and B be integers. Assume that there exists a lottery for at most n processes, on B values, with the property that for every number of processes $1 \leq t \leq n$, and every participating process P_i, with probability at least $1/f(t)$ the maximal drawn number was drawn by the process P_i and all other participating processes draw strictly smaller numbers. Then, there exists a randomized mutual-exclusion algorithm for n processes, that guarantees f-fairness, and uses a shared-variable of $2 \log B + O(1)$ bits.*

By this theorem, in order to prove the existence of mutual-exclusion algorithms, it is enough to prove the existence of the appropriate lotteries. For example, the lottery used in [Rab82, KR92] assigns a probability of 2^{-i} for each value $1 \leq i < B$ ($B = 4 + \log n$), and probability 2^{-B+1} for the value B. It is shown that this lottery gives $f(t) = O(t)$ and therefore can be used to achieve mutual-exclusion with linear fairness.

Note that all the upper bounds we give, immediately give upper bounds on the number of states of (n, f)-live Markov-chains, for the appropriate f's. We start by showing an upper bound for a constant size shared variable.

Theorem 14: *There exist a randomized mutual-exclusion algorithm that uses a constant size shared variable, and guarantees $1/2^t$-fairness.*

Proof: We show a lottery with $f(t) = 2^t$; by Theorem 13, this completes the proof. In the lottery, each participating process P_i, chooses a value in $\{1, 2\}$ with uniform distribution, i.e. the probability that P_i chooses each of the two values is $1/2$. For every participating process P_i, we are interested in the event that P_i chooses the maximum value and it is unique. Since there are only two possible values, this is simply the event that P_i chooses the value 2 and all other participating processes choose 1. The probability that P_i chooses the value 2 and all other participating processes choose 1 is exactly $1/2^t$, therefore $f(t) = 2^t$. □

The next theorem derives a bound in the case where the fairness guarantee needs to be polynomial in t (note that in the previous theorem the fairness guarantee is exponential in t). We show the result by exhibiting a different lottery for this case. This lottery implies an upper bound of $O(\log \log \log n)$ bits for mutual exclusion with polynomial fairness.

Theorem 15: *For any constants $c > 1$, there exist a randomized mutual-exclusion algorithm that uses an $O(\log \log \log n)$ size shared variable, and guarantees $\Omega(1/t^c)$-fairness.*

The first summand is less than $1/(2t^c)$, by Lemma 7. If the graph of the α-heavy edges is covered and no α-light edge was used, we cannot reach a new state. Therefore, the second summand is not more than the probability of not covering the graph in $t-1$ steps (given that no α-light edge is used). By Lemma 8, this probability is also less than $1/(2t^c)$. All together, we get that the probability of visiting a new state in step t is less than $1/t^c$. This implies that the Markov chain does not have the required liveness property. □

Corollary 10: *Let $c \geq 0$ be any constant. Let $(Q_{i,j})$ be any Markov-chain on k states which is (n, t^c)-live. Then, $k = \Omega(\sqrt{\log\log n / \log\log\log n})$.*

Proof: Lemma 6 shows that if there is no "gap" of the form $(\alpha^{\lambda k}, \alpha]$ then the claimed lower bound holds. Lemma 9 shows that if there is such a "gap" then the Markov-chain is not (n, t^c)-live. □

Theorem 11: *Every mutual-exclusion algorithm for n processes which guarantees t^c-fairness requires a shared variable of $\Omega(\log\log\log n)$ bits.*

Proof: The proof is the similar to the proof of Theorem 4, but using Corollary 10 instead of Lemma 3. Consider the algorithm $\mathcal{A}$ and the corresponding Markov-chain $Q(\mathcal{A}, d)$, and assume that the algorithm $\mathcal{A}$ guarantees t^c-fairness. If the Markov-chain $Q(\mathcal{A}, d)$ is not $(n-1, 2t^c)$-live, then this implies that there exists a $1 \leq t \leq n-1$ which is $\frac{1}{2t^c}$-good. By Lemma 2, there exists an extension of the basic schedule such that the probability of P_t to enter the critical section at either C_1 or C_2 is less than $\frac{1}{t^c}$, contradicting the t^c-fairness of the algorithm. Therefore, $Q(\mathcal{A}, d)$ must be $(n-1, 2t^c)$-live. By Corollary 10, this implies that k, the number of states in this Markov-chain, is $\Omega(\sqrt{\log\log n / \log\log\log n})$. By the construction of the Markov-chain, the number of bits in the shared variable used by $\mathcal{A}$ is $\log(k-1)$ which is therefore $\Omega(\log\log\log n)$, as claimed. □

To relax the requirement that the processes have the same program we make the following observations: For every process P_i, we can associate with its program $\mathcal{A}_i$, a Markov-chain $Q(\mathcal{A}_i, d)$, as before. All those Markov-chains have the same number of states k. If $k = \Omega(\sqrt{\log\log n / \log\log\log n})$ we are done. That is, the number of bits of the shared variable is $\Omega(\log\log\log n)$. By Lemma 6, if k is "too small" (i.e., $k = o(\sqrt{\log\log n / \log\log\log n})$), then for every process there is some gap. That is, one of the k^2 intervals $(\beta_{\ell+1}, \beta_\ell]$ considered in the proof of Lemma 6 is empty. Moreover, for n/k^2 of the processes the gap is in the *same* interval. Denote this interval by $(\alpha^{\lambda k}, \alpha]$. Now, consider only these processes and the α-heavy edges in the corresponding graphs. The number of ways of choosing for each of the k^2 edges whether it is heavy or not is 2^{k^2}. Therefore, there are $n' \stackrel{\triangle}{=} n/(k^2 \cdot 2^{k^2})$ processes with the same gap, and the same α-heavy edges. If we take only these processes, the same proof can be repeated. Finally, note that due to the double logarithmic relation between k and n, the number of processes we remained with is

$$n' = n/(k^2 \cdot 2^{k^2}) > n^{\varepsilon},$$

for some constant $\varepsilon > 0$. Hence, we get a lower bound of $\Omega(\log\log\log n') = \Omega(\log\log\log n)$ also for the case that processes may use different programs. To conclude,

Theorem 12: *Every mutual-exclusion algorithm for n processes which guarantees t^c-fairness requires a shared variable of $\Omega(\log\log\log n)$ bits, even if each process runs a different program.*

Lemma 8: *Consider a Markov chain $(Q_{i,j})$ such that each transition probability is either α-heavy or α-light. The probability that after a walk W of t steps, which uses only α-heavy edges, the induced (directed) graph of α-heavy edges is not completely covered is less than $1/(2t^c)$.*

Proof: Recall that k is the number of states in the Markov-chain. We divide the walk W into $b = \lceil t/k \rceil$ blocks of size k. Consider the location v of the walk at the beginning of a block. Either all the nodes reachable from v in the induced graph of α-heavy edges were already visited; or, there is some node v', reachable from v, which was not visited yet. This implies that there is a (simple) path of length at most k from v to v' consisting of α-heavy edges (there may be more than one such path; however, we cannot make any stronger assumption, e.g., the existence of an α-heavy edge connecting v to v'). Therefore, the probability that the walk visits v' during the current block of steps is at least α^k.

By standard Chernoff bounds, the probability that the graph is not completely covered after t/k blocks is at most $e^{-\frac{t\alpha^k}{8k}}$. To see this, define a random variable X_i which is 1 if the graph is either completely covered by the first $i-1$ blocks of the walk, or a new node is visited during the i-th block of the walk. Otherwise, $X_i = 0$. By the above, the probability of X_i to be 1 is at least $p = \alpha^k$. Let S be the sum of these random variables. That is, $S \stackrel{\triangle}{=} \sum_{i=1}^{b} X_i$. With this definitions, the event $S \geq k$ implies that the graph is completely covered. The Chernoff bound shows that $Pr(S \leq (1-\varepsilon)pb) \leq e^{-bp\varepsilon^2/2}$, which for $\varepsilon = 1/2$ and our choices of b and p gives $e^{-\frac{t\alpha^k}{8k}}$. Finally, note that $(1-\varepsilon)pb = (1/2)\alpha^k t/k$ which is greater than k for our choice of t. Therefore, $Pr(S < k) \leq Pr(S < (1-\varepsilon)pb) \leq e^{-\frac{t\alpha^k}{8k}}$.

Finally, we need to show that $e^{-\frac{t\alpha^k}{8k}} < 1/(2t^c)$. It is enough to show that $-\frac{t\alpha^k}{8k} < -\ln(2t^c)$ or that $2t > \frac{16k}{\alpha^k} c \ln(2t)$. It can be easily verified that for $D \geq 3$ the equation $x > D \ln x$ is true for any $x \geq D^2$. In our case we take $D = \frac{16ck}{\alpha^k}$ (note that $c > 1, k \geq 1$ and $\alpha < 1$ hence indeed $D \geq 3$). Therefore, the choice of t (with γ sufficiently large) guarantees the inequality. The lemma follows. □

Given that the walk does not use any α-light edge, and since every edge is either α-light or α-heavy, the probability that in step t the walk visits a state in which it already visited is at least the probability that a walk of length $t-1$ completely covers the induced graph of α-heavy edges (since by the definition of "completely covered", the only nodes that can be reached in step t, by an α-heavy edge, have been already visited).

Lemma 9: *Consider a Markov chain $(Q_{i,j})$ such that each transition probability is either α-heavy or α-light. For any $c > 1$, the Markov-chain is not (n, t^c)-live.*

Proof: In order to show that the Markov-chain is not (n, t^c)-live, it is sufficient to show that there exists a t, such that the probability that in step t a new state is visited is at most $1/t^c$. The probability of reaching a new state at step t is,

$$\begin{aligned}
& Pr(\text{new state in step } t) \\
= \; & Pr(\text{new state in step } t | \alpha\text{-light edge is used}) \cdot Pr(\alpha\text{-light edge is used}) \\
+ \; & Pr(\text{new state in step } t | \text{no } \alpha\text{-light edge is used}) \cdot Pr(\text{no } \alpha\text{-light edge is used}) \\
\leq \; & Pr(\alpha\text{-light edge is used}) + Pr(\text{new state in step } t | \text{no } \alpha\text{-light edge is used})
\end{aligned}$$

Lemma 6: *Let $\lambda > 1$ be a constant.*[10] *Let $(Q_{i,j})$ be a Markov-chain over k states. If for every $0 \leq \alpha \leq 1/2$, such that $\alpha^{\lambda k} \geq 1/n$, there exist i and j, such that $Q_{i,j} \in (\alpha^{\lambda k}, \alpha]$, then $k = \Omega(\sqrt{\log\log n / \log\log\log n})$.*

Proof: Consider the sequence $\beta_\ell = 2^{-(\lambda k)^\ell}$ $(\ell = 0, 1, \ldots)$. By the assumption, if $\beta_{\ell+1} \geq 1/n$ then the interval $(\beta_{\ell+1}, \beta_\ell]$ contains at least one of the values $Q_{i,j}$ (also note that these intervals are disjoint). Since there are at most k^2 such values, then $\beta_{k^2+1} < 1/n$, otherwise not all the intervals contain a value $Q_{i,j}$. From this inequality we get that $k = \Omega(\sqrt{\log\log n / \log\log\log n})$. □

In the following we assume that there is such a gap, i.e. there exists an $\alpha \leq 1/2$ such that the interval $(\alpha^{\lambda k}, \alpha]$ contains no probability $Q_{i,j}$, and $\alpha^{\lambda k} \geq 1/n$. An edge $i \to j$ with probability $Q_{i,j} \leq \alpha^{\lambda k}$ is called α-light, otherwise if $Q_{i,j} > \alpha$ it is called α-heavy. The assumption that there is a gap implies that every edge is either α-heavy or α-light. We consider a random walk of (a suitably chosen) length $t < n$. We show two main properties. The first is that the probability that in t steps of the Markov-chain some α-light edge is used is "small". The second is that the probability that we do not "cover" the graph induced by the α-heavy edges is "small". Before going into the details, we will make our choice of parameters, as follows:

$$\begin{aligned} t &= \frac{\gamma k^2 c^2}{\alpha^{2k}} \qquad \text{and} \\ \lambda &= \gamma' c \log c, \end{aligned}$$

where γ and γ' are sufficiently large constants, and k is the number of states. (Unfortunately, the best intuition that we can give for the choice of t and λ is that they make the proof go through.) We start by showing that the probability of traversing some α-light edge is negligible.

Lemma 7: *The probability that any α-light edge is used in a walk of length t is less than $1/(2t^c)$.*

Proof: In each of the t steps, the walk can choose among at most $k-1$ α-light edges, each with probability at most $\alpha^{\lambda k}$. Therefore, the probability that any α-light edge is used is not more than $t \cdot k \cdot \alpha^{\lambda k}$. To see that this is less than $1/(2t^c)$, it is sufficient to show that $2t^{c+1} k \alpha^{\lambda k} < 1$. We now substitute the value of t into this inequality and we get that it is sufficient to prove $2\gamma^{c+1} k^{2c+3} c^{2c+2} \alpha^{(\lambda - 2(c+1))k} < 1$. As $\alpha < 1/2$ it is enough that $2\gamma^{c+1} k^{2c+3} c^{2c+2} < 2^{(\lambda-2(c+1))k}$. This is satisfied as long as $(\lambda - 2(c+1))k > 1 + (c+1)\log\gamma + (2c+3)\log k + (2c+2)\log c$. Hence, choosing λ as above, with γ' sufficiently large, will satisfy the inequality, and the lemma follows. □

In the following we define what it means to cover a directed graph. Intuitively, a directed graph is covered by a walk if no new node can be reached.

Definition 2: *A directed graph is* completely covered *by a walk W, if each node that is reachable from the last node of the walk W has been already visited in W.*

Note that the above definition does *not* require that the walk visits *all* the nodes in the graph, just that there are no *new* nodes which can be reached from the last node. The next lemma gives a bound on the probability that we completely cover the graph induced by the α-heavy edges.

[10] the value of λ depends on the value of the constant c.

Theorem 4: *Every mutual-exclusion algorithm $\mathcal{A}$ for n processes which guarantees $O(t)$-fairness requires a shared variable of $\Omega(\log\log n)$ bits.*

Proof: Consider the algorithm $\mathcal{A}$ and the corresponding Markov-chain $Q(\mathcal{A}, d)$, and assume that the algorithm $\mathcal{A}$ guarantees ct-fairness. If the Markov-chain $Q(\mathcal{A}, d)$ is not $(n-1, 2ct)$-live, then there exists a t, $1 \leq t \leq n-1$ which is $\frac{1}{2ct}$-good. By Lemma 2, there exists an extension of the basic schedule (in which P_n was scheduled d times) such that the probability of P_t to enter the critical section at either C_1 or C_2 is less than $\frac{1}{ct}$, contradicting the ct-fairness of the algorithm. Therefore, $Q(\mathcal{A}, d)$ must be $(n-1, 2ct)$-live. By Lemma 3, this implies that k, the number of states in this Markov-chain, is at least $\frac{1}{2c}\ln(n-1)$. By the construction of the Markov-chain, the number of bits in the shared variable used by $\mathcal{A}$ is $\log(k-1)$ which is therefore at least $\log(\frac{1}{2c}\ln(n-1)) = \Omega(\log\log n)$, as claimed. □

Note that in the proof of Lemma 3 we do not use the fact that in each step we use the same transition matrix. In other words, the lemma holds even if we associate with every step i a different transition matrix $Q^{(i)}$. This implies that the lower bound of Theorem 4 still holds even if the processes are allowed to use different programs.

Corollary 5: *Every mutual-exclusion algorithm for n processes which guarantees $O(t)$-fairness requires a shared variable of $\Omega(\log\log n)$ bits, even if each process runs a different program.*

In the above discussion, we assumed that the adversary knows what is the value t which is $\frac{1}{2ct}$-good. We can make this assumption because the adversary is given the algorithm $\mathcal{A}$ and he knows the number of steps d, taken by P_n before entering the CS phase. Therefore, he can construct the above Markov-chain. Based on this, the adversary can compute the probability of visiting a new state at any given step, and hence finding what the value of t is.

4 Lower Bound: Non-Linear Fairness

In this section we extend the results from the case of linear fairness to the case of polynomial (t^c, for $c > 1$) fairness. The proof goes along the same lines, except that the proof of Lemma 3 fails in this case, since $\sum_i 1/i^c = O(1)$, for $c > 1$. Thus, a different approach is required.

To simplify the proof, we assume that all the processes are identical (i.e. both code and initial state). At the end of the section we show that the proof can be extended to the case that the processes are not identical. Our goal now is to derive a lower bound for the number of states of (n, t^c)-live Markov-chains.

Consider the k^2 values $Q_{i,j}$ of the Markov chain. We divide the proof into two cases according to the way these values are distributed in the interval $[0, 1]$. The easy case is when these values are "dense" in the interval. Lemma 6 below claims that in this case k must be "large". Then, we handle the more difficult case where there exists some "gap" in the interval $[0, 1]$ in which none of these k^2 values fall, and show that in this case the Markov-chain is not (n, t^c)-live.

(where d will be taken as the number of times P_n was scheduled before entering the critical section; this parameter is known to the adversary). Note that d does depend on $\mathcal{A}$ but for each specific d we have a different Markov-chain. Recall that we assume, at this point, that all the processes run the same program.

States – The states of the Markov-chain correspond to the possible values of the shared-variable. In addition there is a special initial state q_0 (i.e., if we have a k-bit shared-variable then the Markov-chain has $2^k + 1$ states).

Transition probabilities – For $i, j \geq 1$, the entry $Q_{i,j}$ equals the probability that a process, when invoked for the first time (i.e., it is in its initial state) and reading the value i from the shared-variable writes the value j. This probability is defined by the algorithm $\mathcal{A}$. For the initial state and $i > 0$, we define $Q_{0,i}$ to be the probability that the process P_n, before closing the critical section, wrote the value $v[n] = i$ (this probability depends on d!). Also, $Q_{i,0} = 0$ for all i.

The idea is that the behavior of a process which is scheduled to read the shared-variable for the first time depends only on the current value of the shared-variable and does not depend on the whole history of values. This Markov property enables us to describe the process of writes as a Markov chain. The relation between this Markov-chain and the schedule we are constructing is formalized by the following claim; later we concentrate on analyzing the Markov-chain.

Claim 1: *Fix a schedule as above. Also, let $\mathcal{A}$, d and $Q(\mathcal{A}, d)$ be as above. Then, for every sequence of values $V_n, V_1, \ldots, V_i$,*

$$Pr[(s_0 = V_n) \wedge (s_1 = V_1) \wedge \ldots \wedge (s_i = V_i)] \quad = \quad Pr[(v[n] = V_n) \wedge (v[1] = V_1) \wedge \ldots \wedge (v[i] = V_i)],$$

where $s_0, s_1, s_2, \ldots$ is the sequence of states visited by the Markov-chain.

Proof: The proof follows by an easy induction from the definition of the Markov-chain. □

It follows from the definitions that if every t is not $1/ct$-good then the corresponding Markov-chain is $(n, c \cdot t)$-live, hence if the Markov-chain is not (n, ct)-live then there exists a t which is $1/ct$-good. (Recall that a Markov Chain $(Q_{i,j})$ is $(n, f(t))$-*live* if for every $1 \leq t \leq n$ the probability that the t^{th} step reaches a state that was not visited during the first $t-1$ steps is at least $1/f(t)$.) The following lemma gives a bound on the number of states of *any* Markov-chain (not only those constructed as above) with linear liveness property.

Lemma 3: *Let $c \geq 0$ be any constant. Let $(Q_{i,j})$ be any Markov-chain on k states which is $(n, c \cdot t)$-live. Then, $k \geq \frac{1}{c} \ln n$.*

Proof: For every $1 \leq i \leq n$, let X_i be a random variable which takes the value 1 if the Markov-chain visits a new state in its i^{th} step and 0 otherwise. Clearly, $\sum_{i=1}^{n} X_i$ is at most the number of states k, and hence also $E[\sum_{i=1}^{n} X_i] \leq k$. By linearity of expectation, $E[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} E[X_i]$. By the liveness of the Markov-chain, for every i, $Prob[X_i = 1] \geq 1/ci$. This implies that $E(X_i) \geq 1/ci$. Combining all together we get that $k \geq \sum_{i=1}^{n} 1/ci = \frac{1}{c} \cdot \ln n + O(1)$. □

1. Schedule P_n until it enters the critical section.
2. Schedule $P_1, \ldots, P_t$ (each is scheduled once).
3. Schedule P_n until it exists the critical section.
4. For $i = 1, \ldots, t-1$
   Schedule $P_1, \ldots, P_i$ for M_i steps (in a round-robin).
5. Schedule $P_1, \ldots, P_n$ (in a round-robin).

Figure 2: Strategy for the adversary (assuming the existence of t which is δ-good)

Lemma 2: *Given that t is δ-good, there exists an extension of the above schedule such that the probability that P_t enters at either C_1 or C_2 is at most $\delta + \varepsilon$, where ε is arbitrarily small.*

Proof: As t is δ-good, the adversary, who knows the algorithm used, knows that with high probability there exists an s ($1 \leq s < t$) such that $v[t]$ equals $v[s]$. However, the adversary has to overcome the fact that he does not know the value of s. He will do so by trying all the possible values of s. The problem is that trying one value of s, say $s = 9$, influences other values of s, say $s = 3$, since the processes may notice that $s > 3$. The first idea is to try $s = 1, 2, \ldots, t-1$ in this order. This guarantees that before trying $s = i$, the only processes scheduled are $P_1, \ldots, P_{i-1}$ (this is done by modifying step (4); see Figure 2).

Essentially, when the adversary checks whether $s = i$ it does the same thing as in the proof of Lemma 1 above. Namely, it schedules only $P_1, \ldots, P_i$. If indeed $s = i$, then the adversary is guaranteed that if it schedules only $P_1 \ldots P_i$, eventually one of them would enter at C_1 and one at C_2 (it might be the same process in both cases). If the scheduler detects that $s \neq i$, then it continues to $i + 1$. We are guaranteed that, with probability $1 - \delta$, there exists such an i.

We need to show how the adversary can check whether $s = i$ or $s \neq i$. If the adversary has a bound on the number of steps until the processes $P_1, \ldots, P_i$ would let one in C_1 and another in C_2, say M_i steps, it would schedule them this many steps. If no process would enter at either C_1 or C_2, then the scheduler is guaranteed that $s \neq i$. As in the proof of Lemma 1 it may be the case that such bound M_i does not exist. For this reason it would compute the value of M_i such that if $s = i$ the probability that one of $P_1, \ldots, P_i$ enters at C_1 and C_2 during at most M_i steps (in a round robin schedule) is at least $1 - \varepsilon$.

The probability that the scheduler misses the right value of s is ε (note that we do not care about the other cases). In addition, we assumed that there exists such an s, with probability $1 - \delta$. Therefore, the probability that P_t enters is at most $\delta + \varepsilon$. This is since this probability is bounded by the probability that there is no such s (bounded by ε), plus the probability that, given that there is such an s, the scheduler misses it (bounded by δ). □

So far, we proved that if there is a t which is δ-*good* then the adversary can discriminate against P_t. We now prove that such a t must exist, if the number of values is "too small". At this point it is convenient to define the Markov-chain $Q(\mathcal{A}, d)$ corresponding to a mutual-exclusion algorithm $\mathcal{A}$, and an integer d

1. Schedule P_n until it enters the critical section.
2. Schedule $P_1, \ldots, P_s, \ldots, P_t$ (each is scheduled once).
3. Schedule P_n until it exists the critical section.
4. Schedule $P_1, \ldots, P_s$ for M steps (in a round-robin).
5. Schedule $P_1, \ldots, P_n$ (in a round-robin).

Figure 1: Strategy for the adversary (assuming the existence of s and t as in Lemma 1)

show that if the shared-variable is "too small" then a *good* t must exist. To introduce the idea of the proof we first assume that t satisfies an even stronger property, as formalized in the next lemma.

Lemma 1: *Assume that there exists a specific s, $1 < s < t$, such that the probability that the value $v[t]$ written by P_t into the shared variable equals the value $v[s]$ written by P_s is at least $1-\delta$. Then, there exists an extension of the above schedule such that the probability that P_t enters at either C_1 or C_2 is at most $\delta + \varepsilon$, where ε is arbitrarily small.*

Proof: Assume that $v[t] = v[s]$. That is, P_t wrote into the shared-variable the same value as P_s (this happens with probability at least $1-\delta$). The adversary extends the schedule by first scheduling P_n to access the shared-variable until it moves to the REMAINDER phase and the critical section is open (Figure 1 step (3)). This is guaranteed by the fault-freeness property. Then (Figure 1 step (4)), the adversary continues by scheduling only P_1 to P_s (say, by a round robin). Observe that the only way that a process P_i can note that another process P_j was scheduled before him is if P_j changed the value of the shared-variable. Hence, if indeed $v[t] = v[s]$, then in this case processes P_1 to P_s must operate as if P_{s+1} to P_t were not scheduled. This implies (by the Deadlock freeness property, and the first part of the Fairness property) that eventually some process P_i enters at C_1 and some other process P_j at C_2, where $1 \leq i, j \leq s$. More precisely, there exists a large enough M, such that if $P_1, \ldots, P_s$ are scheduled to take M steps then with probability at least $1 - \varepsilon$ two of these processes enter at time C_1 and time C_2 (if this does not happen during the M steps this could be either because P_t did *not* write the same value as P_s or because none of $P_1, \ldots, P_s$ entered the critical section). Hence, the probability that P_t enters at either C_1 or C_2 is bounded by the probability that it did not write the same value as P_s plus the probability that M steps were not enough for $P_1, \ldots, P_s$, which is at most $\delta + \varepsilon$. □

The problem with the above lemma is that the adversary needs to know some fixed s, such that the probability that P_t writes to the shared variable the same value that was written by P_s, is "high". The next lemma shows that the same bound holds even in the case that s is not fixed. First we define the notion of *good* t.

Definition 1: *We say that t is δ-good[9] if the probability that the value $v[t]$ written by P_t into the shared variable equals one of $v[1], \ldots, v[t-1]$ (the values written by $P_1, \ldots, P_{t-1}$ respectively) is at least $1 - \delta$.*

[9]The term "good" is from the adversary point of view.

In the definition of f-Fairness, we require that a process that tries to enter at time C_i will have a "good" chance to enter at the *next* time, i.e. C_{i+1}. At first sight, it seems more natural to require that a process that arrives between time C_{i-1} and C_i will have a good probability to enter at C_i, as defined in [Rab82]. However, as pointed out by [Sai92], such a statement is circular as the definition of the event C_i depends on whether the process enters the critical section or not, and it seems that there is no "acceptable" way to get around this problem. We follow here the solution suggested by [KR92], that requires the "good" chance to be only in the next time step.

2.2 Markov Chains

Let $S = \{s_1, s_2, \ldots, s_k\}$ be a set of k states. A *Markov-chain* is a real, non-negative, $k \times k$ matrix $(Q_{i,j})$ with the property that the sum of elements in each row equals 1. It can be used to generate sequences of elements of S in the following way. Start in the initial state, say, s_1. At each step, if the last element in the sequence is s_i, move to state s_j with probability $Q_{i,j}$. (i.e., the probability of moving into state s_j depends only on the last state s_i and not on the whole history).

We say that a Markov Chain $(Q_{i,j})$ is $(n, f(t))$-*live* if for every $1 \leq t \leq n$ the probability that the sequence generated by the above process visits in the t^{th} step a state that was not visited during the first $t-1$ steps is at least $1/f(t)$.

It is sometimes convenient to think about the Markov-chain as a complete directed graph on k nodes. Every edge $i \to j$ has a value $Q_{i,j}$ which is the probability of visiting node j in the next step when being in node i. The sequence of states, in this case, is usually called a *walk*.

3 Lower Bound: Linear Fairness

In this section we describe the lower bound for the case of *linear fairness*; that is, where the probability of each process to enter the critical section is required to be inversely proportional to the number of processes trying to enter the critical section. To do so we describe a strategy for the adversary scheduler, given a mutual-exclusion algorithm $\mathcal{A}$, to plan a schedule in which the probability of a certain process (that the adversary wish to discriminate against) to enter the critical section in a given round is smaller than what is required.

The schedule starts by scheduling the process P_n to access the shared-variable until this process enters to CS phase; i.e., the critical section is *closed* (Figure 1 step (1)). The deadlock-freeness property guarantees that this will eventually happen. Denote by d the number of steps taken by P_n before entering the CS phase, and by $v[n]$ the value that P_n wrote into the shared variable at this time. Then, the adversary schedules each of the processes P_1 to P_t (in order) to perform a single read-modify-write operation on the shared variable, where t is a parameter (Figure 1 step (2)). We denote by $v[i]$ the value written by P_i into the shared variable. The proof of the lower bound has two parts. We first show that if t is *good* (in a sense that will be defined later) then the adversary can discriminate against P_t; namely, with high probability, process P_t will not enter the Critical Section (although scheduled to access the shared-variable). Later, we

shared variable nor the content of any local variable.[5] More formally, let a *run* be a (finite or infinite) sequence $(i_1, x_1), \ldots, (i_k, x_k), \ldots$, where x_j indicates which phase process P_{i_j} started or whether it accessed the shared variable. A run is called *proper* if the subsequence of phases corresponds to every process P_i is of the form: REMAINDER, TRYING, CS, EXIT, REMAINDER,...

A *scheduler* is a (probabilistic) function that on a finite run σ gives the identity of the next process to access the shared variable. It should satisfy the following property:

Scheduler-Liveness: For each time t, and any process P_j not in REMAINDER phase at time t, there exists a time $t' > t$ in which P_j makes a move.

Next, we discuss the correctness conditions of a randomized mutual-exclusion algorithm. While the first three conditions are rather standard, the fairness definition is the one unique to the randomized solutions. These correctness conditions may be generalized in various ways without affecting the results. We discuss possible extensions of the conditions throughout the paper.

Mutual-exclusion: At any time t there is at most one process in CS phase. If there is a process in CS phase then we say that the critical section is *closed*, otherwise, it is *open*.

Fault freeness: If a process P_j moves from TRYING phase to CS phase then eventually P_j moves from CS phase to EXIT phase and from EXIT phase to REMAINDER phase. (E.g., a protocol in which a process after entering the critical section gets into an infinite loop violates this condition.)

Deadlock freeness: If the critical section is *open* and there is a process in TRYING phase, then eventually some process enters to CS phase[6].

f-Fairness: Let C_i be the time at which the ith closing of the critical section occurs. Let S_i be the set of processes in TRYING phase, that were scheduled to access the shared-variable between time C_{i-1} and C_i, excluding the process that entered the critical section at C_i.

1. If $S_i \neq \emptyset$ then one of the processes in S_i enters the critical section at C_{i+1}.
2. For every process $P_j \in S_i$, the probability that P_j enters at time C_{i+1} is at least $1/f(t)$, where $t = |S_i|$[7].

In particular, for a constant c, we refer to $c \cdot t$-Fairness as *Linear* Fairness, and to t^c-Fairness as *Polynomial* Fairness. [8]

[5] note that since we are interested in this work in proving *lower* bounds, this assumption makes our job more complicated.

[6] The definition can be weakened to require that this will hold with probability one, and all the results of the paper will remain valid.

[7] The probability space is defined on prefixes of runs; therefore, the space is finite. Formally, the above requirement says that for any prefix of a run up to C_i, σ, which have a non-zero probability, the probability that P_j enters at time C_{i+1} given σ is at least $1/f(t)$. Later when we will refer to an event as "happened in the past" we would mean that it is satisfied by σ.

[8] It is possible to relax the definition of fairness and allow each party that was scheduled to access the shared-variable between time C_{i-1} and C_i, and not entered the critical section at times $C_i, C_{i+1}, \ldots, C_{i+d-1}$, for some parameter d, to compete on entering at time C_{i+d} with probability of success at least $1/f(t)$. The results and the proofs (with few minor changes) hold for such a definition as well.

linear fairness.

We define a slightly weaker fairness property that we term *polynomial fairness*: If a process participates in a round together with m processes, it has a probability of $\Omega(1/m^{1+\varepsilon})$ to enter the critical section in the next round (where $\varepsilon > 0$ is a constant). Surprisingly, we show that for every $\varepsilon > 0$ an $O(\log\log\log n)$-bit shared-variable is sufficient to achieve polynomial fairness, and that $\Omega(\log\log\log n)$-bit shared-variable is necessary. Hence, this slight weakening of the fairness property results in an exponential reduction in the size of the required shared-variable. Finally, we show that with a constant size shared variable it is possible to guarantee an $\Omega(1/2^m)$ probability of entering the critical section.

For our lower bound proofs, we study general *Markov Chains*; i.e., those which are represented by a $k \times k$ real non-negative matrix where the sum of elements in each row is 1. We call a Markov-chain *live* if in each of its first n steps it has a "considerable probability" of visiting a new state (i.e., a state that was not visited in each of the previous steps). We relate the fairness of mutual exclusion algorithms to the liveness of Markov-chains (where the different notions of fairness correspond to different interpretation of "considerable probability"). We obtain our bounds for mutual exclusion by proving bounds on k, the number of states of the Markov chains (as a function of n). We believe that our bounds and technique may be found useful for other applications.

The paper is organized as follows. In section 2 we give formal definition of the mutual-exclusion problem and of Markov-chains. In Sections 3 and 4 we prove the lower bounds for linear/polynomial fairness (respectively). Finally, Section 5 includes the upper bounds.

2 Preliminaries

2.1 Mutual Exclusion

In this section we define the properties required from a randomized mutual exclusion algorithm. Let $P_1, \ldots, P_n$ be the n processes in the system. The processes coordinate their activities by using a shared read-modify-write variable v. (In addition, each process P_i has unbounded local memory). While it is convenient to assume that all the processes run the same program, our results do not depend on this assumption. During the computation, each process P_i is in one of four possible phases: *TRYING phase*, in which it attempts to enter the critical section, *CS (Critical Section) phase*, in which it executes the critical section, *EXIT phase*, in which it leaves the critical section, or *REMAINDER phase*, in which it does other local computations.

At any given time the adversary scheduler[4] can observe the *external behavior* of the processes (i.e., which of the four phases each process currently executes), and use this information (together with its information on the past behavior of the processes) to determine which process will be the next to access the shared variable. (It is also assumed that the adversary knows the algorithm used by the processes, including the initial state of each process.) The adversary scheduler *cannot* observe the content of the

[4] We assume here the same adversary scheduler and the same correctness conditions as in [KR92].

determined by a scheduler.

The mutual exclusion problem is a classical problem in distributed computing. It was first suggested by Dijkstra [Dij65], who solved the problem using a (one-bit) semaphore. While the semaphore does solve the problem and guarantees *deadlock freedom*, it does not guarantee any *fairness* among the processes competing for the critical section; a process that is waiting for the critical section may wait forever. Since then, numerous solutions were proposed for the mutual-exclusion problem. All these solutions guarantee *deadlock freedom*, together with some notion of *fairness*.

An important parameter for evaluating the complexity of a mutual exclusion algorithm is the size of the shared variable that is used. As mentioned above, to guarantee only *deadlock freeness*, a one-bit semaphore is sufficient [Dij65]. Burns et. al. [BFJ+82] define the *bounded-waiting* property as a fairness criterion. Roughly speaking, this property guarantees that between the first time that a process accesses the shared-variable in order to try to enter the critical section and the time it actually enters the critical section, each of the other processes may enter the critical section at most once. They proved that if *deterministic* algorithms are used then an $\Omega(\log n)$-bit shared variable is required for achieving bounded-waiting, and that this number of bits is also sufficient.

Rabin [Rab82] suggested the use of *randomized* algorithms for mutual exclusion, and defined the notion of fairness for such algorithms. Roughly speaking, the fairness of a randomized mutual-exclusion algorithm measures the probability of a process to enter the critical section in a given time, as a function of the number of processes concurrently competing for the critical section. Specifically, Rabin was interested in the following fairness property, which we refer to as *linear fairness*: if a process participates in a "round"[3] together with m processes, it has a probability of $\Omega(1/m)$ to enter the critical section in the next round. This property can be considered as a probabilistic analogue of the bounded-waiting property. Randomized algorithms having the linear-fairness property that use $O(\log\log n)$-bit shared variable are presented in [Rab82, KR92]. This is in contrast to the $\Theta(\log n)$-bit shared-variable required by deterministic algorithms.

Proving the correctness of such randomized distributed protocols involves many delicate issues. Saias [Sai92] developed a general methodology to prove the correctness of a randomized distributed protocol. The main difficulty of such proofs is the need to deal with two separate sources of nondeterminism: the randomness that the protocol generates, and the decisions of the adversary. The key idea in his methodology is that these two ingredients should be made independent. Using his systematic methodology, Saias [Sai92] uncovered the flaw in [Rab82].

No *lower bounds* for randomized mutual-exclusion were known. In fact, in light of the results mentioned above, it may seem plausible that a constant size shared-variable is sufficient for mutual-exclusion with linear fairness. More than that, it was shown [Rab82, KR92], that a constant size shared-variable may be powerful; it suffices for guaranteeing that each of the competing processes will have $\Omega(1/n)$ probability to enter the critical section. However, this is *independent* of m and hence is much weaker. In this paper, we prove a tight $\Omega(\log\log n)$-bit lower bound on the size of the shared variable that is needed for achieving mutual-exclusion with linear fairness. Thus, in particular, a constant size shared-variable cannot guarantee

[3] A round is the time between two consecutive closings of the critical section.

For many applications in distributed environments, there is a provable gap between the power of randomized algorithms and their deterministic counterparts. The most renowned example is achieving Byzantine agreement with linear number of faults; while any deterministic algorithm requires at least a linear number of rounds [FL82], there is a randomized algorithm that performs the same task in a constant number of rounds [FM88]. Another important example is that of reaching a consensus in an asynchronous distributed system with faults: this is impossible with deterministic protocols, even if the faults are restricted to a single fail-stop fault [FLP85], but is possible with the use of randomized protocols (see [CIL87]).

The gap between the performance of randomized and deterministic algorithms exists also for the *mutual exclusion* problem. The complexity measure here is the size of the shared variable.[1] Any deterministic algorithm requires an $\Omega(\log n)$ bit shared-variable, in order to achieve mutual exclusion (with fairness) between n distinct process, and this bound is tight [BFJ+82]. On the other hand, there is a randomized algorithm requiring only an $O(\log\log n)$ bit shared-variable [Rab82, KR92].[2]

It remained an open problem whether the complexity of the randomized algorithm for the mutual-exclusion problem can be farther reduced, perhaps even to a constant number of bits. A constant size shared-variable is of special interest, since it implies that the size of the shared memory can be independent of the number of the processes using it. Our main contribution is a tight $\Omega(\log\log n)$ lower bound on the number of bits required to implement the shared variable. Other tight *upper* and *lower* bounds are given for mutual-exclusion with weaker fairness properties.

Few lower bounds are known for randomized distributed algorithms. Many of these lower bounds are based on arguments that arise from the need of information to flow from one side of the network to the other side, or based on the symmetry between different processes [IR81]. Another type of argument for randomized lower bounds is through the use of the min-max theorem [Yao77, AS91]. For randomized Byzantine agreement, in the case that more than a third of the processes are faulty, a lower bound on the success rate is known [KY84, GY89].

Previous Work and Our Results

In the following we give a more detailed description of the mutual exclusion problem and a summary of our results together with previous results related to our work.

The setting of the *mutual-exclusion* problem is as follows. There are n processes that from time to time need to execute a critical section in which exactly one is allowed to employ some shared resource. The processes can coordinate their activities through a shared *read-modify-write* variable (i.e., reading and re-writing the shared variable is an atomic action). The sequence of accesses to the shared-variable is

[1] Throughout this work the size of the shared variable is measured in terms of the number of *bits* of the shared-variable rather then the number of *values* (as is done in some of the papers in the literature). Clearly, the number of bits is logarithmic in the number of values, hence a constant factor in the number of bits translates to a polynomial factor in the number of values. Still, the number of bits is a very natural measure for the size of variables.

[2] The first solution for this problem was given in [Rab82]. A flaw in this solution was pointed out by [Sai92]. A new solution, based on ideas of [Rab82], was given in [KR92].

Lower Bounds for Randomized Mutual Exclusion *

Eyal Kushilevitz† Yishay Mansour‡ Michael O. Rabin§ David Zuckerman¶

Abstract

We establish, for the first time, lower bounds for *randomized* mutual-exclusion algorithms (with a read-modify-write operation). Our main result is that a constant size shared-variable cannot guarantee strong fairness, even if randomization is allowed. In fact, we prove a lower bound of $\Omega(\log\log n)$ bits on the size of the shared-variable, which is also tight.

We investigate weaker fairness conditions and derive tight (upper and lower) bounds for them as well. Surprisingly, it turns out that slightly weakening the fairness condition results in an exponential reduction in the size of the required shared-variable. Our lower bounds rely on an analysis of Markov-chains, that may be of interest on its own and may have applications elsewhere.

Keywords: Mutual Exclusion, Randomized Distributed Algorithms, Markov-Chains, Lower-Bounds.

1 Introduction

Randomization has played an important role in the design and understanding of distributed algorithms. It is a natural tool which is usually used in order to break symmetry between identical processes in a distributed system. Beyond its natural role in symmetry breaking, randomization often increases the computation power (e.g., [LR81, Ben83]), significantly decreases computational costs (e.g., [Bra85, FM88]), and helps in simplifying algorithms.

*An early version of this paper appeared in *Proc. of 25th ACM Symp. on Theory of Computation*, May 1993, pp. 154-163. Research of Eyal Kushilevitz and Michael Rabin supported by research contracts ONR-N0001491-J-1981 and NSF-CCR-90-07677 at Harvard University.

†Research was done while the author was at Aiken Computation Lab., Harvard University. Current address: Dept. of Computer Science, Technion. e-mail: eyalk@cs.technion.ac.il .

‡Computer science Dept., Tel-Aviv University and IBM – T. J. Watson Research Center. e-mail: mansour@math.tau.ac.il

§Aiken Computation Lab., Harvard University and Institute of Mathematics, Hebrew University of Jerusalem. e-mail: rabin@das.harvard.edu .

¶Lab. for Computer Science, MIT. e-mail: diz@theory.lcs.mit.edu . Supported by an NSF Postdoctoral Fellowship.