Contention-free Complexity of Shared Memory Algorithms *

Rajeev Alur† Gadi Taubenfeld‡

Abstract

Worst-case time complexity is a measure of the maximum time needed to solve a problem over all runs. Contention-free time complexity indicates the maximum time needed when a process executes by itself, without competition from other processes. Since contention is rare in well-designed systems, it is important to design algorithms which perform well in the absence of contention. We study the contention-free time complexity of shared memory algorithms using two measures: step complexity, which counts the number of accesses to shared registers; and register complexity, which measures the number of different registers accessed. Depending on the system architecture, one of the two measures more accurately reflects the elapsed time.

We provide lower and upper bounds for the contention-free step and register complexity of solving the mutual exclusion problem as a function of the number of processes and the size of the largest register that can be accessed in one atomic step. We also present bounds on the worst-case and contention-free step and register complexities of solving the naming problem. These bounds illustrate that the proposed complexity measures are useful in differentiating among the computational powers of different primitives.

*An abbreviated version of this paper appeared in the *Proceedings of the 13th Annual ACM Symposium on Principles of Distributed Computing* (PODC 1994) pp. 61–70, and an extended version appeared in *Information and Computation*, vol. 126(1), pp. 62–73, 1996.

†Bell Laboratories, Lucent Technologies, 700 Mountain Avenue, Murray Hill, NJ 07974, USA. alur@research.bell-labs.com.

‡AT&T Bell Laboratories and Israel Open University. gadi@research.att.com.

1 Introduction

Worst-case versus contention-free complexity

We consider an asynchronous distributed environment where processes communicate via shared registers. For such shared memory systems, there has been extensive research aimed at understanding the time needed to solve a variety of coordination and synchronization problems, such as; mutual exclusion, consensus, and renaming. Most of this research has focused on the worst-case complexity, which indicates the maximum time taken by any process to accomplish its task. The *contention-free* complexity, on the other hand, gives an upper bound on the time used by a process when the process runs by itself without any interference from other processes. Since contention is rare in many well-designed systems, it is important to design algorithms which perform well *also* in the absence of contention. Thus, between two algorithms with the same worst-case complexity, the one with the lower contention-free complexity may perform better in practice. The importance of contention-free complexity was first explored by Lamport [Lam87], who presented a mutual exclusion algorithm in which any process accesses shared registers only a constant number of times before entering the critical section, in the absence of contention.

Unlike the worst-case complexity, there are no known tight bounds on the contention-free complexity of concurrent tasks. The proofs of lower bounds on the worst-case complexity usually involve constructing runs with many processes interfering with each other, and do not imply lower bounds on the contention-free complexity. This means that it may be possible to improve existing algorithms by modifying them to have better contention-free complexity. This calls for research with the goal of obtaining tight bounds on contention-free complexity, and to understand the relationship between contention-free and worst-case complexity for a variety of problems. This paper initiates such an investigation by presenting bounds for the contention-free time complexity for mutual exclusion in a shared memory with atomic registers, and for the naming problem in a shared memory that supports different types of read-modify-write bits.

Measures of time complexity

There is no agreement on the appropriate measure of time complexity in asynchronous shared-memory systems. Most suggestions agree that the time it takes to access the shared memory is the dominating factor, since this is typically much slower then the time required for local computation. The traditional measure of time complexity accounts for the total number of accesses to the shared registers. We define the *step complexity* of a task to be maximum number of times any process accesses shared registers. For each problem, the worst-case and contention-free step complexities can be defined in the obvious way. For instance, the worst-case step complexity of an algorithm for solving consensus is the maximum number of times any process accesses shared registers, from its start until it decides.

The definition of step complexity does not always capture the real time complexity of an algorithm [PF77]. If one process is in a loop waiting for another process to complete some action, then increasing the execution speed of the first process increases its number of steps, without necessarily degrading the total system performance. This leads to the following alternative definition (from [PF77]). Assume that every step of every process takes at most 1 time unit. Thus, if two processes start at the same time and one takes 100 steps while the other takes 5 steps, then the total time elapsed is at most 5 units. Time complexity is then defined by counting time units instead of steps. For our investigation, it is enough to notice that in the absence of contention, this last complexity measure and step complexity are the same.

Recently, there have been many suggestions to measure time complexity differently. Yang and Anderson propose to distinguish between a *remote* and a local access to shared memory [YA93]. Shared registers may be locally accessible as a result of coherent caching, or when using distributed shared memory where memory is physically distributed among the processors. The definition of a remote access is very delicate and depends on specific architectural details of a given system (see [YA93] for one possible definition). Assuming that no process physically owns any portion of the shared memory, it follows that the first access to a new shared register by a process must be a remote access (since the process does not yet have a copy of this register in its cache). Thus, the number of different registers accessed by a process in a given run is a lower bound on the number of its remote accesses. We call this measure *register complexity*.

As in step complexity, for each problem, we can define worst-case register complexity and contention-free register complexity. For instance, the contention-free register complexity of a consensus algorithm is the maximum number of different registers accessed by a process along runs in which, while this process is executing, all other processes have either decided, or failed, or not started. Observe that in the contention-free runs, once a process obtains a local copy, there is no other process executing concurrently that can invalidate the local copy, and hence, it is reasonable to assume that the number of different registers accessed accurately reflects the number of remote accesses.

It is clear that register complexity is a lower bound for step complexity. Depending on the architecture, one of the two measures more accurately reflects the time taken. It is also worth noting that register complexity is different from the space complexity, which is usually defined to be the total number of shared registers used in the algorithm.

Finally, note that the definitions of (worst-case) step and register complexity are different from the definition of time complexity explored by Dwork *et al.* [DHW93], which accounts for the contention level – the number of processes that access the same shared memory simultaneously. For example, if n processes are trying to access the same memory location simultaneously, then according to our measures, it will take each one of them one step to do it. According to [DHW93], it will take the i'th process i steps because it must wait for the other $i-1$ processes to finish.

Bounds for mutual exclusion

The mutual exclusion problem is to design a protocol that guarantees mutually exclusive access to a critical section among a number of competing processes [Dij65]. We assume that the shared memory supports only atomic registers. The contention-free step (register) complexity of a mutual exclusion algorithm is measured by counting the number of accesses to the shared registers (the number of different shared registers accessed) in the entry code and the exit code, when no other processes are competing. Lamport presented a mutual exclusion algorithm for n processes that has constant contention-free step complexity

[Lam87]. However, these accesses involve reading and writing registers of size $\log n$ bits.

An interesting question is whether it is possible to further improve Lamport's solution by reducing the number of times shared bits need to be accessed. In particular, is there an algorithm in which the number of times a process only needs to access a constant number of shared bits before entering its critical section, in the absence of contention? Our first result implies that no such algorithm exists. We show that the contention-free step complexity of every mutual exclusion algorithm for n processes is greater than $\frac{\log n}{\ell-2+3\log\log n}$, where ℓ is the size (in terms of bits) of the biggest register accessed by the algorithm in one atomic step. We also prove a lower bound of $\sqrt{\frac{\log n}{\ell+\log\log n}}$ on the contention-free register complexity. This implies that the contention-free register complexity cannot be only a constant number of bits. For upper bounds, we give an algorithm with contention-free step as well as register complexity $O(\lceil\frac{\log n}{\ell}\rceil)$.

In modern multiprocessors, it is possible to load or store 32-bit or even 64-bit words, so having registers of size $\log n$ bits poses no problem. However, although our study concerns registers of smaller size, and is mainly of theoretical interest, it may be of practical value as well. Some recent shared-memory systems support access to words of memory at different granularities (i.e., multi-grain atomic reads and writes). Thus, several registers of smaller size can be packed into one word of memory, enabling reads or writes to all or a subset of them in one atomic step. This was demonstrated by Michael and Scott [MS93], who improve the performance of Lamport's algorithm [Lam87] and the authors' algorithm [AT92] by more than 25%, by exploiting the ability to read and write atomically at both full- and half-word granularities.

Note that in the worst-case, a process may be forced to take an unbounded number of steps before entering its critical section; that is, the worst-case step complexity of mutual exclusion is infinity [AT92]. The algorithm presented by Kessels, [Kes82], which is based on the idea of using a tournament tree from [PF77], uses only shared bits and has $O(\log n)$ worst-case register complexity. No lower bound is known for the worst-case register complexity for large registers.

Yand and Anderson independently study the time complexity of mutual exclusion [YA94]. The results imply (1) an $\Omega(\log_w n)$ lower bound on the contention-free step complexity and an $\Omega(\sqrt{\log_w n})$ lower bound on the contention-free register complexity, where w is the maximum number of processes that may simultaneously write the same register, and (2) an $\Omega(\log_c n)$ lower bound on the contention-free register complexity, where c is the maximum number of processes that may simultaneously access the same register. In another related paper [BL93], it is shown that any deadlock-free mutual exclusion algorithm for n processes in an asynchronous system must use n shared registers.

Bounds for naming

In Section 3, we consider the naming problem. The goal of the naming problem is to assign unique names to a collection of initially identical processes. The processes may crash (i.e. stopping failures), and the solution is required to be wait-free, that is, it should guarantee that every participating process will always be able to terminate in a finite number of steps, regardless of the behavior of other processes [PF77, Her91]. We consider a shared memory system that supports atomic access only to individual bits, and consider a variety of models corresponding to different combinations of allowed operations, such as; test-and-set, read, and test-and-flip. (This last operation flips the bit and returns the old value, and is similar to the balancer of [AHS91].)

Our results give tight lower and upper bounds on worst-case and contention-free, step and register complexities of naming, in some of these models. These results, admittedly not of direct practical interest on their own, demonstrate the differences between contention-free versus worst-case complexity, and between step complexity versus register complexity. For instance, if the model supports only the test-and-set operation, $n-1$ is a tight bound on all four measures. Strengthening the model with a read operation lowers both measures of the contention-free complexity to $\log n$. Introducing test-and-reset to this model lowers the worst-case register complexity to $\log n$, whereas with the availability of test-and-flip operation, $\log n$ is a tight bound on all four measures. Thus, a consideration of all four measures of time complexity brings out the differences in the computational powers of different primitives.

2 Mutual exclusion

2.1 Problem definition

The mutual exclusion problem is to design a protocol that guarantees mutually exclusive access to a critical section among a number of competing processes [Dij65]. The solution to the problem should satisfy: (a) no two processes are in their critical section at the same time (*mutual exclusion*), and (b) if some process p starts executing its algorithm, then eventually some process (possibly different from p) is in its critical section (*deadlock freedom*). A weaker version of the problem replaces the requirement of deadlock freedom by *weak deadlock freedom* which requires deadlock freedom only in the absence of contention.

We consider the mutual exclusion problem in a shared memory environment that supports only atomic registers, that is, a process can only either read or write a shared register in one atomic step. We assume that there are n competing processes, where each process is assumed to have a unique identifier from $\{1, ..., n\}$. We define the atomicity of an algorithm to be the size (in terms of bits) of the biggest register accessed by the algorithm in one atomic step. We will use ℓ to denote the atomicity of an algorithm.

2.2 Complexity measures

We outline our formal model of distributed systems, followed by definitions of the complexity measures. Processes are modeled as (possibly infinite) state-machines, communicating by writing and reading from shared registers.

A (global) *state* of the system includes the state of each process and the values of all shared registers. An *event* is a single step of some process, and is either an access to a shared register, or simply an update of the internal state of a process.

As in the standard interleaving semantics, a *run* σ of the system is an alternating sequence $s_0 \xrightarrow{e_0} s_1 \xrightarrow{e_1} \cdots$ of states s_i and events e_i such that (1) the initial state s_0 satisfies the initial condition, and (2) every state s_{i+1} is derived from the previous state s_i by executing the event e_i. Each event belongs to a unique process. The notion of a run, as defined here, captures the asynchronous nature of a system, where no assumptions are made about the relative speeds of the processes.

For a run σ and indices $i \leq j$, the run fragment σ_{ij} denotes the subsequence $s_i \xrightarrow{e_i} \cdots \xrightarrow{e_{j-1}} s_j$ of σ. The step complexity of a process p in a run fragment σ_{ij} is the number of events in σ_{ij} that involve access to shared registers, and belong to p. The register complexity of a process p in a run fragment σ_{ij} is the number of different shared registers accessed by p in this run fragment.

The worst-case step (register) complexity of an algorithm for mutual exclusion gives an upper bound on the time spent (the number of registers accessed) by the winning process in its entry code and exit code. In the formal definition we give below, for simplicity it is assumed that a process does not take any steps in its critical section. Formally, the worst-case step (register) complexity of the entry code of an algorithm is the maximum step (register) complexity of a process p in a run fragment $\sigma_{ij} = s_i \xrightarrow{e_i} \cdots \xrightarrow{e_{j-1}} s_j$ such that

1. process p is in its entry code in state s_i, and is in its critical section in state s_j, and
2. no process is in its exit code or critical section in states s_k for $i \leq k < j$.

The second condition ensures that we start counting the number of steps (registers) of the winning process p only after the processes previously in the critical section have finished their exit code. The worst-case step (register) complexity of the exit code of an algorithm is the maximum step (register) complexity of a process p in a run fragment σ_{ij} such that process p is in its critical section in state s_i, and in its exit code in state s_j. The worst-case step (register) complexity of an algorithm is the sum of the worst-case step (register) complexity of its entry code and exit code.

The contention-free step (register) complexity of an algorithm is the maximum step (register) complexity of a process p in a run fragment σ_{ij} such that

1. process p is in its entry code in state s_i, and is in its exit code in state s_j, and
2. in all states s_k, $i \leq k \leq j$, process p is not in its remainder region, and all other processes $p' \neq p$ are in their remainder regions.

It is known that the worst-case step complexity of mutual exclusion is infinite [AT92]. The worst-case register complexity can be as low as $\log n$, even when atomicity is only 1 (i.e., all shared registers are bits) [Kes82], and there is no known matching lower bound. We consider bounds for contention-free complexity. In our results, sometimes we make finer distinctions such as read-step complexity and write-step complexity (or read-register, write-register) to refer separately to the number of read and write accesses. As usual, the complexity of a problem is defined as the complexity of the best algorithm to solves it.

2.3 The contention detection problem

The lower bounds are proven for a much weaker problem, namely, the problem of detecting contention. The *contention detection problem* requires that when a process is activated, it executes its protocol, and terminates with some output value from $\{0, 1\}$ such that

- in every run, at most one process terminates with output value 1, and

- in a run in which only one process p is activated, p terminates with output 1.

The contention detection problem can be viewed as a single-shot mutual exclusion problem (that is, a process entering its critical section stays there forever), where only weak deadlock freedom is required as a liveness property. A solution to the mutual exclusion problem can be used to solve the contention detection problem as follows. A process entering its critical section sets some special bit to 1, and outputs 1, and every other process that notices that the bit is set to 1, returns 0. This implies that

Lemma 1 *A lower bound on any time complexity (worst-case or contention-free, step or register) of the contention detection problem is also a lower bound on the corresponding time complexity of the mutual exclusion problem.*

2.4 Lower bound on contention-free step complexity

Let $\mathcal{A}$ be an algorithm for detecting contention. For each process p, let $run(p)$ denote the (finite) sequence of accesses to the shared registers by process p in a run where only process p is activated. Let $W(p,m)$ be the pair (x,v), where x is a shared register and v is a value, if the m-th write operation along $run(p)$ is $write(x,v)$ ($W(p,m)$ is undefined if no such operation exists). If $W(p,m)$ is (x,v), we will also write $W^r(p,m) = x$ and $W^v(p,m) = v$. For a process p, let $R(p)$ be the set of shared registers x such that $run(p)$ contains the operation $read(x)$. We first prove the following lemma.

Lemma 2 *Let $\mathcal{A}$ be an algorithm for detecting contention, and let p_1 and p_2 be distinct processes. Then for some m, $W(p_1,m)$ and $W(p_2,m)$ are well defined, with $W(p_1,m) \neq W(p_2,m)$, and either $W^r(p_1,m) \in R(p_2)$ or $W^r(p_2,m) \in R(p_1)$.*

Proof: Consider two distinct processes p_1 and p_2. Suppose each of the runs $run(p_1)$ and $run(p_2)$ contain w write operations. (If the two runs do not contain the same number of operations, then we can introduce some dummy write operations at the end of one of the runs.) The proof will be by contradiction. Assume that for all $1 \leq m \leq w$, if $W(p_1,m) \neq W(p_2,m)$, then $W^r(p_1,m) \notin R(p_2)$ and $W^r(p_2,m) \notin R(p_1)$. We will merge the runs $run(p_1)$ and $run(p_2)$ to obtain a run in which both p_1 and p_2 terminate with output value 1, violating the safety requirement of the problem.

We will inductively construct the runs $\sigma_0, \sigma_1, \ldots \sigma_w$, such that at the end of each σ_i, for each process p_j, for $j = 1, 2$, the following conditions hold: (i) the sequence of operations by process p_j in σ_i is precisely the prefix of $run(p_j)$ up to the i-th write operation, and (ii) for each $x \in R(p_j)$, the value of x is either its initial value or the value written by p_j in its last write operation to x. Thus at the end of each σ_i, p_1 is "hidden" from p_2 and vice versa. Let σ_0 be the empty run. Now assume that we have a run σ_i that satisfies both the conditions (i) and (ii), and we describe how to extend it to obtain σ_{i+1}. First, let p_1 execute read operations until it is ready to execute its $(i+1)$-th write operation. Then let p_2 execute its read operations and also its $(i+1)$-th write operation. Now let p_1 execute its $(i+1)$-th write operation. Either both processes write the same value to the same register, or each process writes to some register that is never read by the other one. Then σ_{i+1} satisfies the requirements (i) and (ii). Thus, we can construct inductively the run σ_w in which both processes terminate with output value 1, a contradiction. ∎

Now we prove the following lemma, which relates the number of processes, atomicity, the number of write operations, and the number of different registers read.

Lemma 3 *Let $\mathcal{A}$ be an algorithm for detecting contention among n processes, with atomicity ℓ, contention-free read-register complexity r, and contention-free write-step complexity w. Then,*

$$w \cdot \ell + w \cdot \log(w^2 \cdot r + w \cdot r^2) \geq \log n.$$

Proof: Let $\mathcal{A}$ be the given algorithm for detecting contention among a set P of n processes. For each $p \in P$, $run(p)$ contains at most w write operations, and the size of $R(p)$ is at most r. We will construct a sequence $P_0, P_1, \ldots$, each $P_i \subseteq P$, and a sequence $W_0, W_1, \ldots$, each $W_i \subseteq \{1, \ldots w\}$ such that for each i,

1. $n \leq (2^\ell \cdot w \cdot r \cdot (w + r))^i \cdot |P_i|$,
2. $|W_i| = i$, and
3. for all $p, p' \in P_i$ and $j \in W_i$, $W(p, j) = W(p', j)$.

Initially, let P_0 be P and $W_0 = \emptyset$. Now, given P_i and W_i satisfying the above 3 conditions, we show how to obtain P_{i+1} and W_{i+1}. Let $|P_i| = n_i$. First, consider the following definitions.

- A register x is a *read-witness* for a process $p \in P_i$, if $x \in R(p)$, and there exists a set $P' \subseteq P_i$ such that $|P'| \geq \frac{n_i}{w+r}$, and for each $p' \in P'$, there exists $j \notin W_i$ with $W^r(p', j) = x$.
- A register x is a *write-witness* for a process $p \in P_i$, if $W^r(p, j) = x$ for some $j \notin W_i$, and there exists a set $P' \subseteq P_i$ such that $|P'| \geq \frac{n_i}{w+r}$, and for each $p' \in P'$, $x \in R(p')$.

From Lemma 2, and the assumption that condition 3 is satisfied, we get that for every distinct $p, p' \in P_i$, there exists $j \notin W_i$ such that either $W^r(p, j) \in R(p')$ or $W^r(p', j) \in R(p)$. Each process reads at most r registers and writes at most w registers. Hence, each $p \in P_i$ has either a read-witness or a write-witness. It follows that either some process has a read-witness or all processes have write-witnesses. Next we prove that in either case there exists a register x and a set $P' \subseteq P_i$ of processes such that $|P'| \geq \frac{n_i}{r(w+r)}$ and for each process $p' \in P'$ there exists $j \notin W_i$ such that $W^r(p', j) = x$.

- Case 1: There exists a process $p \in P_i$ that has a read-witness. Let x be the read-witness for p. By definition, there is a set $P' \subseteq P_i$ such that $|P'| \geq \frac{n_i}{w+r}$ and for each process $p' \in P'$ there exists $j \notin W_i$ such that $W^r(p', j) = x$.
- Case 2: Each process $p \in P_i$ has a write-witness. Each write-witness is read by $\frac{n_i}{w+r}$ number of processes. Since each process reads at most r registers, there are at most $n_i \cdot r$ number of different read operations along the (contention-free) runs of processes in P_i. This implies that at most $r \cdot (w+r)$ different registers can be write-witnesses. Since each process has a write-witness, it follows that some register x is a write-witness for at least $\frac{n_i}{r(w+r)}$ processes. That is, there is a set $P' \subseteq P_i$ such that $|P'| \geq \frac{n_i}{r(w+r)}$ and for each process $p' \in P'$ there exists $j \notin W_i$ such that $W^r(p', j) = x$.

Since each run has at most w write operations, it follows that there exists $j \notin W_i$ and a set $P'' \subseteq P'$ of size at least $\frac{n_i}{rw(w+r)}$ such that for each $p' \in P''$, $W^r(p', j) = x$. Let $W_{i+1} = W_i \cup \{j\}$. Since there are only 2^ℓ different values that can be written to a register, there exists a value v, and a set $P_{i+1} \subseteq P''$ of size at least $\frac{n_i}{2^\ell rw(w+r)}$ such that for each process $p' \in P_{i+1}$, $W(p', j) = (x, v)$. Thus, we have the desired sets P_{i+1} and W_{i+1}.

Since each run contains at most w write operations, from Lemma 2, it follows that the set P_w can contain at most one process. This implies that $n \leq (2^\ell \cdot w \cdot r \cdot (w+r))^w$, and hence $\log n \leq w \cdot \ell + w \cdot \log(w^2 \cdot r + w \cdot r^2)$. ∎

The proof of this lemma views a run of the processes as a set of registers (corresponding to read operations) and a sequence of pairs of registers and values (corresponding to write operations). Lemma 2 states some properties of such runs using the problem requirements, whereas the proof of Lemma 3 is purely combinatorial: it involves only a counting argument for runs satisfying Lemma 2.

Lemma 4 *Let $\mathcal{A}$ be an algorithm for detecting contention among n processes, with atomicity ℓ and contention-free step complexity c. Then,*

$$c > \frac{\log n}{\ell - 2 + 3 \log \log n}.$$

Proof: Let r be the contention-free read-step complexity, and let w be the contention-free write-step complexity. Since c denotes the contention-free step complexity, by definition,

$$c \geq w + r. \tag{1}$$

Before a process terminates, it must read (and write) at least once. That is $r \geq 1$ (and $w \geq 1$) and hence,

$$c - 1 \geq w. \tag{2}$$

From Lemma 3,

$$w \cdot \ell + w \cdot \log(w^2 \cdot r + w \cdot r^2) \geq \log n. \tag{3}$$

Using the above inequalities, we will find a good approximation for c. From (1), $w + r = c'$ for some $c' \leq c$. The expression $w^2 \cdot r + w \cdot r^2$ gets its maximum value when $w = r = c'/2$, and hence it is bounded from above when w and r are assigned the value $c/2$. That is, for $w + r \leq c$,

$$\frac{c^3}{4} = \left(\frac{c}{2}\right)^2 \cdot \frac{c}{2} + \frac{c}{2} \cdot \left(\frac{c}{2}\right)^2 \geq w^2 \cdot r + w \cdot r^2. \tag{4}$$

From (3) and (4), it follows that,

$$w \cdot \ell + w \cdot \log \frac{c^3}{4} \geq \log n. \tag{5}$$

Clearly, the left-hand side of inequality (5) has maximum value when w is maximum. Hence by (2), we can substitute $c - 1$ for w to obtain

$$(c-1) \cdot \ell + (c-1) \cdot \log \frac{c^3}{4} \geq \log n, \tag{6}$$

which is the same as,

$$(c-1)\cdot\log(2^{\ell-2}\cdot c^3) \geq \log n. \tag{7}$$

To solve (7), we use the following simple fact. For positive k and m,

$$\text{if} \quad c\cdot\log(k\cdot c^3) \geq m \quad \text{then} \quad c > \frac{m}{\log(k\cdot m^3)}. \tag{8}$$

Using (8), we get from (7) that,

$$c > \frac{\log n}{\ell - 2 + 3\log\log n}. \tag{9}$$

▮

From Lemma 1 and Lemma 4, we get the following lower bound for mutual exclusion:

Theorem 1 *Let $\mathcal{A}$ be a (weak) deadlock-free mutual exclusion algorithm for n processes, with atomicity ℓ and contention-free step complexity c. Then,*

$$c > \frac{\log n}{\ell - 2 + 3\log\log n}.$$

One immediate consequence is that, although the contention-free step complexity may be a constant when the atomicity is big enough, the number of times a process needs to access shared bits cannot be a constant, regardless of the level of atomicity. More precisely, in every algorithm with atomicity ℓ and contention-free step complexity c, the number of times some process needs to access shared bits in the absence of contention is at least $\ell + c - 1$.

2.5 Lower bound on contention-free register complexity

Now we consider the contention-free register complexity. Since it is possible to access the same register several times, Theorem 1 says nothing about the number of different registers that must be accessed. However, it is possible to prove that there is no algorithm in which only a constant number of distinct shared bits are accessed by a process before entering its critical section, in the absence of contention.

Let $\mathcal{A}$ be an algorithm for detecting contention among n processes. Let c be the contention-free register complexity of $\mathcal{A}$, and let w be the contention-free write-register complexity of $\mathcal{A}$. For each process p, let $run(p)$ denote the sequence of accesses to the shared registers by process p when only p is activated. Let $wr(p)$ denote the sequence of registers written by p along $run(p)$ such that a register x appears before a register y along $wr(p)$ iff along $run(p)$, the first write operation to x precedes the first write operation to y. The length of $wr(p)$ is bounded by w, and for the worst-case analysis in our proofs, it suffices to assume that the length of $wr(p)$ equals w for every process p. For $1 \leq m \leq w$, let $wr(p, m)$ denote the m-th register of the sequence $wr(p)$.

The contention-free run $run(p)$ can be divided in $w+1$ stretches as follows. Suppose $wr(p)$ is the sequence $x_1 x_2 \ldots x_w$. The 0-th stretch contains only read operations. The first stretch starts with a write operation to x_1, and contains the subsequence of $run(p)$ up to the first write to x_2. Thus, all of its write operations involve x_1. The second stretch starts with a write operation to x_2, and contains writes to the registers x_1 and x_2. Thus, at the beginning of the i-th stretch, p is about to write to x_i, and all registers, except $x_1, \dots x_{i-1}$, have their initial values. Observe that the i-th stretch contains arbitrarily many read and write operations, but its every write involves one of the registers $x_1, \dots x_i$. Let $wv(p, i, j)$ denote the value of x_j at the beginning of the i-th stretch. That is, $wv(p, i, j)$ is the initial value of x_j if $i \leq j$, and if $j < i$ then it is the last value written by p to x_j before its first write to x_i. For a worst-case analysis, it suffices to assume that the i-th stretch contains at least one write operation to each of the registers $x_1, \dots x_i$. Thus, each process p in its i-th stretch writes $wv(p, i, j)$ to x_j for $j < i$.

For a set P of processes, let *shared*(P) be the set of indices $1 \leq m \leq w$ for which there are two distinct processes $p, p' \in P$ such that $wr(p, m)$ is read by p' along $run(p')$. Let *common*(P) be the set of indices $1 \leq m \leq w$ such that for all $p, p' \in P$, $wr(p, m) = wr(p', m)$ and $wv(p, m', m) = wv(p', m', m)$ for all $1 \leq m' \leq w$.

Lemma 5 *Let P be a set of processes such that shared(P) = common(P). Then $|P| < 2 \cdot w!$.*

Proof: Let P be a set of n processes. For the worst-case analysis, assume that *shared*(P) = *common*$(P) = \{1, \dots w\}$. Hence, for all $p, p' \in P$, $wr(p') = wr(p)$, let this sequence of registers be $x_1 \dots x_w$. Also for all $p, p' \in P$, and $1 \leq m, m' \leq w$, $wv(p', m, m') = wv(p, m, m')$. We will denote the value of x_j at the beginning of the i-th stretch by v_j^i. (All processes in P agree on these values by assumption.)

We combine the contention-free runs of the processes in P. We inductively construct the runs $\sigma_1, \sigma_2, \dots \sigma_w$, such that at the end of each σ_i the following conditions hold:

1. there is a set P_i of $n/i!$ processes such that for every $p \in P_i$, the sequence of operations by p in σ_i is the same as the prefix of $run(p)$ containing the first $(i-1)$ stretches,
2. for every $j < i$, there is a set P_j^i of $n/i!$ processes such that each $p \in P_j^i$ is about to write the value v_j^i to the register x_j, and
3. σ_i does not contain any write operations to the registers $x_i, \dots x_w$.

Constructing the run σ_1 is easy: simply let each process $p \in P$ execute its 0-th stretch one after the other. (Recall that the 0-th stretch contains only read operations.) The set P_1 equals P.

Now suppose we have constructed the run σ_i satisfying the above three conditions. Each process in P_i is about to execute its i-th stretch. For a process $p \in P_i$ to be able to execute its i-th stretch, it is required that the value of each register x_j, for $j \neq i$, be v_j^i. The value of x_i does not matter because p is about to write to x_i. Each process in P_j^i is about to write v_j^i to x_j for $j < i$. For $j > i$, the register x_j has not yet been changed, and its value at the end of σ_i is same the as its initial value, which also equals v_j^i. Hence, we can let one process from each of the sets P_j^i, $j < i$, execute one write operation, and then let a process in P_i execute its i-th stretch (note that $|P_i| = |P_j^i|$, for each $j < i$).

We let $|P_i|/(i+1)$ processes execute their i-th stretch to completion, and this constitutes the set P_{i+1} of processes ready to execute the next stretch. For each $j \leq i$, each process writes the value v_j^{i+1} to the register x_j along its i-th stretch (it's the last value it writes to

x_j along this stretch). Hence, for each $j \leq i$, we let $|P_i|/(i+1)$ processes execute their i-th stretch only partially, and suspend these processes just before they are about to write the value v_j^{i+1} to the register x_j. This gives us the sets P_j^{i+1}. The resulting run is the desired σ_{i+1}.

If $n \geq 2w!$, then at the end of σ_w we have $|P_w| \geq 2$. Each process is about to write a value to x_w, and for each $j < w$, we have two processes about to write the value v_j^w to x_j. Hence, as before, we can let both processes in P_w execute their last stretch, violating the safety requirement. ∎

Next we use the above lemma to obtain a relationship among the number of processes, the contention-free register complexity, and atomicity.

Lemma 6 *Let $\mathcal{A}$ be an algorithm for detecting contention among n processes, with atomicity ℓ, contention-free write-register complexity w, and contention-free register complexity c. Then,*

$$n \;<\; 2w!\cdot(4c\cdot w!)^c\cdot(w\cdot 2^{\ell w})^w.$$

Proof: Let $\mathcal{A}$ be the given algorithm for detecting contention among a set P of n processes. We construct a sequence $P_0, P_1, \ldots$ of processes, a sequence $R_0, R_1, \ldots$ of shared registers, and a sequence $W_0, W_1, \ldots$ of sets of indices, each $W_i \subseteq \{1, \ldots w\}$, such that for each i,

1. $n \leq (4c\cdot w!)^i\cdot(w\cdot 2^{\ell w})^{|W_i|}\cdot|P_i|$,
2. $|R_i| + |W_i| = i$,
3. for all $x \in R_i$, for each $p \in P_i$, x is read, but not written, by p along $run(p)$, and
4. $W_i \subseteq common(P_i)$.

Initially, let P_0 be P and $R_0 = W_0 = \emptyset$. Now, given P_i, R_i, and W_i satisfying the above conditions, we show how to obtain P_{i+1}, R_{i+1}, and W_{i+1}. Let $|P_i| = n_i$.

First we construct an edge-labeled (undirected) graph G_i as follows. The graph contains one node for every process p in P_i. Let x be a shared register such that $x \notin R_i \cup \{wr(p,m) \mid p \in P_i \text{ and } m \in W_i\}$. There is an edge between p and p' labeled with x if both p and p' access x along their contention-free runs.

Let Q be an independent set of vertices (i.e. if G_i has an edge between p and p' then at least one of p or p' is not in Q). From the construction it follows that, for $1 \leq m \leq w$, if $m \in shared(Q)$ then $m \in W_i$, and hence, $m \in common(Q)$. From Lemma 5, it follows that $|Q| < 2w!$.

Since the graph G_i does not contain an independent set of size $2w!$, it must contain a vertex p with degree at least $n_i/(2w!)$. The process p accesses at most c registers in its contention-free run, and hence, the edges incident on p have at most c different labels. Thus, there is a register x, and a set Q_1 of processes such that each process in Q_1 accesses x along its contention-free run, and $|Q_1| \geq n'$ where $n' = n_i/(2c\cdot w!)$. Let $Q_2 \subseteq Q_1$ be the set of processes $q \in Q_1$ such that q writes to x along $run(q)$. There are two cases to be considered.

- Case 1: $|Q_2| < n'/2$. We choose P_{i+1} to be the set $Q_1 - Q_2$, R_{i+1} to be the set $R_i \cup \{x\}$, and $W_{i+1} = W_i$. Each process $q \in P_{i+1}$ reads x without writing to x along $run(q)$. Thus condition 3 is satisfied for the register x. $|P_{i+1}| > n'/2$, where $n' = n_i/(2c \cdot w!)$, implying condition 1.

- Case 2: $|Q_2| \geq n'/2$. For each $q \in Q_2$, x appears along $wr(q)$. The length of $wr(q)$ is w, hence, there is a subset $Q_3 \subseteq Q_2$ for which there exists $1 \leq m \leq w$ such that $wr(q, m) = x$ for all $q \in Q_3$, and $|Q_3| \geq |Q_2|/w$. For each $q \in Q_3$, for each j, $wv(q, j, m)$ can have 2^ℓ different values. Hence there is a subset $Q_4 \subseteq Q_3$ of processes such that for all $q, q' \in Q_4$ and for all $1 \leq j \leq w$, $wv(q, j, m) = wv(q', j, m)$, and $|Q_4| \geq |Q_3|/2^{\ell w}$. Thus, $m \in common(Q_4)$. Setting $P_{i+1} = Q_4$, $R_{i+1} = R_i$, and $W_{i+1} = W_i \cup \{m\}$ satisfies all the conditions.

Since the contention-free register complexity is c and contention-free write-register complexity is w, we have $|R_i \cup W_i| \leq c$ and $|W_i| \leq w$. Thus the above procedure can be repeated at most c times. Furthermore, from Lemma 5, $|P_c| < 2w!$. This implies

$$n \;<\; 2w! \cdot (4c \cdot w!)^c \cdot (w \cdot 2^{\ell w})^w.$$

∎

Now the above lemma can be used to prove a lower bound on the contention-free register complexity.

Theorem 2 *Let $\mathcal{A}$ be an algorithm for solving the contention detection problem, or for (weak) deadlock-free mutual exclusion. Let n be the number of processes, ℓ be the atomicity, and c be the contention-free register complexity. Then,*

$$c \geq \sqrt{\frac{\log n}{\ell + \log\log n}}.$$

Proof: From Lemma 6, we get

$$n < 2w! \cdot (4c \cdot w!)^c \cdot (w \cdot 2^{\ell w})^w, \tag{1}$$

where w is the contention-free write-register complexity. Since $w! \leq w^w$, we have

$$\ell \cdot w^2 + 1 + 2c + (c+2) \cdot w \cdot \log w + c \cdot \log c \;>\; \log n. \tag{2}$$

Since $c \geq w$, we can substitute c for w in (2) to obtain

$$\ell \cdot c^2 + 1 + 2c + (c+3) \cdot c \cdot \log c \;>\; \log n. \tag{3}$$

It is easy to check that

$$1 + 2c + (c+3) \cdot c \cdot \log c \leq (c+1)^2 \cdot \log{(c+1)}^2. \tag{4}$$

Thus, putting (3) and (4) together, we get that

$$\ell \cdot c^2 + (c+1)^2 \cdot \log{(c+1)}^2 \;>\; \log n, \tag{5}$$

which gives us

$$(c+1)^2 \cdot \log(2^\ell \cdot (c+1)^2) \; > \; \log n. \tag{6}$$

Using the simple fact, that for every positive x, y and z,

$$\text{if} \quad x \cdot \log(y \cdot x) \geq z \quad \text{then} \quad x \; > \; \frac{z}{\log(y \cdot z)}, \tag{7}$$

it follows immediately from (6) that

$$(c+1)^2 \; > \; \frac{\log n}{\ell + \log\log n}. \tag{8}$$

The bound of the theorem follows, using the fact that c must be an integer. ∎

Even though the lower bound on contention-free register complexity might not be tight, it implies that the total number of different shared bits accessed by a process cannot be constant, even in the absence of contention.

2.6 Efficient contention-free algorithms for mutual exclusion

As far as upper bounds are concerned, for Lamport's fast mutual exclusion algorithm [Lam87], the atomicity is $\log n$, and the contention-free complexity is a constant. For atomicity smaller than $\log n$, we can prove the following result.

Theorem 3 *For every $1 \leq \ell \leq \log n$, there is a deadlock-free mutual exclusion algorithm (and hence also an algorithm for detecting contention) for n processes, with atomicity ℓ, contention-free step complexity $7\lceil \frac{\log n}{\ell} \rceil$, and contention-free register complexity $3\lceil \frac{\log n}{\ell} \rceil$.*

Proof: We use Lamport's fast algorithm as a basic building block. As already mentioned, in this algorithm, in the absence of contention a process needs to access the shared memory five times in order to enter its critical section and twice in order to exit it – a total of seven accesses. Only 3 different registers are accessed. When the atomicity is ℓ, the algorithm can handle 2^ℓ processes.

We design an algorithm for n processes by constructing an 2^ℓ-ary tree where each node is a copy of Lamport's solution for 2^ℓ processes, using a separate set of registers. To enter its critical section process i starts participating in the protocol at leaf $\lceil i/2^\ell \rceil$ (level 1) and advances towards the root of the tree. A process advances to level i if it is a winner at level $i-1$. A process can enter its critical section when it is the winner at the root. To exit its critical section process i simply executes the exit code in all the nodes in its path from the leaf to the root. The depth of the tree is $\lceil \log_{(2^\ell)} n \rceil = \lceil (\log_2 n)/\ell \rceil$, and in the absence of contention a process needs 7 accesses to 3 different registers at each node it visits. The result follows. ∎

The idea of using a binary tree for solving mutual exclusion is due to Peterson and Fischer [PF77]. In the above solution, the number of accesses differ for different values of ℓ. However, the number of times that bits are accessed in the absence of contention is $O(\log n)$, for all values of ℓ. Even though the contention-free time complexity of mutual exclusion is about the same as that of contention detection, their worst-case time complexities are quite different. Contention detection can be solved by an algorithm whose worst-case step complexity is $\lceil \log n/\ell \rceil$. For mutual exclusion for two or more processes, in the presence of contention it is always possible to schedule the processes in a way that will force the winner to busy wait before entering its critical section.

The bounds on the four complexity measures for deadlock-free mutual exclusion for n processes, with atomicity ℓ, are summarized in the following table:

Bounds for Mutual Exclusion		
	lower bound	upper bound
contention-free register	$\sqrt{\frac{\log n}{\ell+\log\log n}}$ (Thm. 2)	$3\lceil \frac{\log n}{\ell} \rceil$ (Thm. 3)
contention-free step	$\frac{\log n}{\ell-2+3\log\log n}$ (Thm. 1)	$7\lceil \frac{\log n}{\ell} \rceil$ (Thm. 3)
worst-case register	$\sqrt{\frac{\log n}{\ell+\log\log n}}$ (Thm. 2)	$O(\log n)$ [Kes82]
worst-case step	∞ [AT92]	∞

3 Naming

In order to further demonstrate the differences among different notions of time complexity, we consider a very simple problem. The *naming* problem is to assign unique names to initially identical processes. A desired property of a solution is that the name space from which the names are assigned is small. A solution to the problem is required to be *wait-free*, that is, it should guarantee that every participating process will always be able to obtain a unique name and terminate in a finite number of steps, regardless of the behavior of other processes (such as abnormal termination).

3.1 Models

For process communication, we use the shared memory model with shared binary registers (bits) initialized to some known values. It is easy to see that if in one atomic step a process can either read or write a shared register, but cannot do both, then the naming problem is not solvable deterministically, since it is not possible to break symmetry. A probabilistic solution for the atomic read and write model is presented in [LP90]. Since we are only interested here in deterministic solutions, we need to make stronger atomicity assumptions. Below we list the eight different operations a process may apply to a single bit. Each operation is defined in terms of how it affects the bit, and whether or not it returns a value.

1. `skip`: has no effect on the value of the bit and does not return a value.
2. `read`: has no effect on the value of the bit, and returns the current value.
3. `write-0`: the value 0 is assigned to the bit, and no value is returned.
4. `test-and-reset`: the value 0 is assigned to the bit, and the old value is returned.
5. `write-1`: the value 1 is assigned to the bit, and no value is returned.
6. `test-and-set`: the value 1 is assigned to the bit, and the old value is returned.
7. `flip`: flips the value of the bit, and does not return a value.
8. `test-and-flip`: flips the value of the bit, and returns the old value.

A *model* is a subset of these eight operations, and it defines the operations that are supported for each shared bit. (Thus, there are 2^8 different models.) For example, in the model {`read, write-0, write-1`}, in one atomic step a process can either read or write a shared bit, but cannot do both. The model which includes all the eight operations is called the *read-modify-write* model in the literature. The `test-and-flip` operation is sometimes called the `fetch-and-complement` operation.

3.2 Complexity measures

As in mutual exclusion, we consider the step and register complexities in the worst and the contention-free cases. The worst-case step (register) complexity of a naming algorithm is the maximum step (register) complexity of a process p in a run fragment $\sigma_{ij} = s_i \xrightarrow{e_i} \cdots \xrightarrow{e_{j-1}} s_j$ such that both the extreme events e_i and e_{j-1} belong to p. For the contention-free case, we take the maximum over run fragments σ_{ij} such that both e_i and e_{j-1} belong to p, and for every process p' either p' has terminated (or failed) in state s_i or p' has not started in state s_j. Thus, in the contention-free case, there is no interference from other processes while p executes; every process either decided (or failed) before p starts, or starts only after p finishes.

Observe that the operations `write-0` and `write-1` are duals of each other, and similarly, the operations `test-and-reset` and `test-and-set` are duals of each other. Two models M and M' are duals of each other if M is obtained by replacing each operation of M' by its dual operation. For the operations `skip, read, flip, test-and-flip`, each operation is its own dual. It is easy to prove that, if a model M is the dual of M' then, for every measure of time complexity, any bounds applicable to M also hold for M'.

3.3 Bounds for naming

In order to further demonstrate the differences among different notions of time complexity, we will find out the contention-free and worst-case, step and register complexities of solving the naming problem in several of the 2^8 possible models. Our goal is not to solve the problem in all models. For most of the models, it is easy to find tight bounds for algorithms that solve the naming problem. We will cover here some of the more interesting models, and leave it as an exercise for the reader to come up with bounds for the other models.

In each of the algorithms below, we assume that initially there are n identical processes, and the algorithms assign the processes unique names from the set $\{1, ..., n\}$. Hence, they are optimal in terms of the size of the name space. The resulting upper bounds are summarized in the following theorem:

Theorem 4

1. *For every model which includes* `test-and-flip`*, there is a naming algorithm whose worst-case step complexity is* $\log n$*.*

2. *For every model which includes both* `test-and-set` *and* `test-and-reset`*, there is a naming algorithm whose worst-case register complexity is* $\log n$*.*

3. *For every model which includes* `test-and-set`*, there is a naming algorithm whose worst-case step complexity is* $n-1$*.*

4. *For every model which includes* `test-and-set` *and* `read`*, there is a naming algorithm whose contention-free step complexity is* $\log n$*.*

Proof: To prove the first part, we design a tree-based algorithm. We use $n-1$ shared bits which are arranged as a balanced binary tree of depth $\log n$. It is assumed that the leaves of the tree are numbered 1 through $n/2$. The initial values of the bits are immaterial. Each process starts at the root, and follows a path to one of the leaves. At each node it applies `test-and-flip` operation to the corresponding shared bit. At an internal node, if the returned value is 0, the process advances towards the left subtree, otherwise, it advances towards the right subtree. At a leaf numbered ℓ, if the returned value is 0, then the process is assigned the name $2\ell-1$, otherwise, it is assigned the name 2ℓ.

To prove the second part, a similar algorithm is used. Again, each process follows a path from the root to a leaf. At each node, since a `test-and-flip` operation is not available, the process alternately applies `test-and-set` and `test-and-reset` operations on the shared bit until either a `test-and-set` operation returns 0 or a `test-and-reset` operation returns 1. The value obtained in the last operation at a node is used, either to proceed to the next level, or to assign a name, as in the first part.

To prove the third part, we design the following trivial algorithm. We use $n-1$ bits with initial value 0, which are numbered 1 through $n-1$. Each process scans the bits, in the same order, starting with the first bit. At each step, the process applies the `test-and-set` operation, and either moves to the next bit if the returned value is 1, or stops when the returned value is 0 or if it has already scanned all the $n-1$ bits. The process is assigned the name equal to the bit on which its (last) `test-and-set` operation returned 0, or the name n, if no operation returned the value 0.

To prove the fourth part, we slightly modify the previous algorithm. Again, $n-1$ bits with initial values 0 are used, which are numbered 1 through $n-1$. Each process starts with a binary search on the array of $n-1$ bits, trying to find the least numbered bit whose value is still 0. Such a search takes exactly $\log n$ steps. All the first $(\log n)-1$ steps are done using read operations only, while the last step of the binary search is a `test-and-set` operation. If the last operation returns 0, then the process stops. Otherwise, it continues as in the algorithm of the previous paragraph, moving on to the next bit when the returned value is 1, or stopping when the returned value is 0 (or if it has already scanned all the $n-1$ bits), The process is assigned the name equal to the bit on which its (last) `test-and-set` operation returned 0, or the name n, if no operation returned the value 0. ∎

Now we consider lower bounds in different models. The lower bound proofs need no assumptions about the size of the name space (it may be infinite), and hence we get stronger results. The next theorem gives a lower bound on all our complexity measures in all models.

Theorem 5 *For every model, the contention-free register complexity of every naming algorithm is no less than* $\log n$*.*

Proof: Let $\mathcal{A}$ be an arbitrary naming protocol. Let σ be a run of $\mathcal{A}$ in which all the processes are scheduled one at a time, one after the other, and when a process is scheduled it runs without interruption until it terminates. Thus, in σ there is no contention, and at any point only one process is participating in the algorithm. Let $R(p, m)$ be the pair (x, v), where x is a shared bit and v is a value, if the m-th operation executed by process p along σ accesses the bit x, and returns the value v. $R(p, m)$ is undefined if p takes less than m steps, and let v be $\perp$ if the m-th operation does not return a value. If $R(p, m)$ is (x, v), we will also write $R^r(p, m) = x$ and $R^v(p, m) = v$. Since all processes are identical, and m-th step of a process is determined by the values returned in earlier steps, we can prove the following: For all processes p, p', and m, if $R(p, m') = R(p', m')$ for all $m' < m$, then (a) $R(p, m)$ and $R(p', m)$ are defined, (b) $R^r(p, m) = R^r(p', m)$, and (c) if $R^r(p, m) = R^r(p, m')$ for some $m' < m$, then $R^v(p, m) = R^v(p', m)$. From these properties, it is straightforward to prove that if there are n processes, then there is some p which takes at least $\log n$ steps. ∎

Recall that register complexity is always bounded above by step complexity, and contention-free complexity is always bounded above by worst-case complexity. Thus the above two theorems imply that for models with `test-and-flip`, $\log n$ is a tight bound on all complexities. The next theorem gives lower bounds on the worst-case step complexity when `test-and-flip` is not available:

Theorem 6 *For every model which does not include* `test-and-flip`*, the worst-case step complexity of every naming algorithm is no less than $n - 1$.*

Proof: We use the following simple observation. If n processes apply the same operation to the same bit, which is different from `test-and-flip`, then the returned values for at least $n-1$ processes must be the same. Now consider the following run. First we activate all the n processes and let each one of them take one step. Since the processes are identical they will apply the same operation to the same bit and hence $n - 1$ of them will get the same response. Thus $n - 1$ processes are still identical (i.e., have the same history). Next we activate these $n - 1$ identical processes and let each one take one step, and are guaranteed to end up with a set of at least $n - 2$ identical processes. We can repeat this procedure $n - 1$ times, and hence at least one process takes $n - 1$ steps. ∎

Note that the above lower bound does not apply to the worst-case register complexity. In fact, from Theorem 4, with `test-and-set` and `test-and-reset`, it can be only $\log n$. It is possible to prove a similar lower bound if the model includes only one of these two operations:

Theorem 7 *For the model* {`test-and-set`}*, the contention-free register complexity of every naming algorithm is no less than $n - 1$.*

Proof: First we observe that with only `test-and-set`, a shared bit can be changed at most once in a given run. We will construct a contention-free run in which processes execute one after the other. Let us order the processes $p_1, \ldots p_n$. First let p_1 execute till it chooses a name, and then let p_2 execute till completion, and so on. It is clear that if p_i does not change any bit, then the next process chooses the same name as p_i does. Hence, each p_i, for $i < n$, changes at least one bit. For $i < n$, let x_i be the first bit that p_i changes (the bits $x_1, \ldots x_{n-1}$ are all distinct). Since all processes are identical, it follows that every process following p_i accesses x_i. It follows that p_{n-1} (as well as p_n) access the $n-1$ distinct registers $x_1, \ldots x_{n-1}$. ∎

Some of our results are summarized in the following table that shows *tight* bounds on various complexity measures for some of the models. (We use c-f and w-c as abbreviations for contention-free and worst-case, respectively.)

Tight Bounds for Naming					
	test-and-set	read + test-and-set	read + test-and-set + test-and-reset	test-and-flip	r-m-w (all)
c-f register	$n-1$	$\log n$	$\log n$	$\log n$	$\log n$
c-f step	$n-1$	$\log n$	$\log n$	$\log n$	$\log n$
w-c register	$n-1$	$n-1$	$\log n$	$\log n$	$\log n$
w-c step	$n-1$	$n-1$	$n-1$	$\log n$	$\log n$

This table illustrates how different synchronization primitives can be distinguished by different complexity measures. If the model supports only `test-and-set` operation, $n-1$ is a tight bound on all four measures. Strengthening the model with `read` lowers both measures of the contention-free complexity to $\log n$. Introducing `test-and-reset` to this model lowers the worst-case register complexity to $\log n$, but the worst-case step complexity still remains $n-1$. With the availability of `test-and-flip` operation, $\log n$ is a tight bound on all four measures. In particular, the traditional measure of worst-case step complexity fails to distinguish between the models of the first three columns. A simultaneous consideration of all four measures of time complexity results in more refined distinctions among the computational powers of these different primitives.

4 Discussion

In this paper we have focused on the notion of contention-free time complexity, which seems to be an important complexity measure in the analysis of distributed algorithms.

Reading from shared-memory can take a long time on today's architectures, compared to reading from a cache. Hence, the time to access the shared memory is strongly influenced by the level of contention – the number of processors that access the same memory location simultaneously. Several formal methods of how to model contention have been suggested in the past, which focus on the worst-case analysis under high level of contention. While good performance in the presence of high contention is important, the behavior in the absence of contention is sometimes even more important. As Lamport points out: "...the current belief among operating system designers is that contention for a critical section is rare in a well designed system" [Lam87]. Thus, "high-contention analysis" may be a misleading indication of performance in practice.

Good time complexity in the absence of contention can help achieve good performance also in the presence of high contention, using a technique called *backoff*: when a process notices contention it delays itself for some time, giving other processes a chance to proceed. Experiments show that in various mutual exclusion algorithms (such as Lamport's fast algorithm) with backoff, regardless of the level of contention, the time it takes the winning process to enter its critical section since the last time a critical section was released, is very close to the time it takes in that absence of contention (for example, see [MS93]).

We have also shown how the size of the biggest register that may be accessed by a process in one atomic step can affect the time complexity of a given problem. Several recent papers have investigated a related question, of how the register size affects the concurrent computability of various problems [AS93, FMRT90, FMT93].

Acknowledgements: We are grateful to Michael Merritt for many helpful discussions, and for carefully proofreading the paper.

References

[AS93] Y. Afek, and G. Stupp. Synchronization power depends on the register size. In *Proc. of the 34th IEEE Ann. Symp. on Foundation of Computer Science*, pages 196–205, 1993.

[AHS91] J. Aspnes, M. Herlihy, and N. Shavit. Counting networks and multi-processor coordination. In *Proc. 23rd ACM Symp. on Theory of Computing*, pages 348–358, May 1991.

[AT92] R. Alur and G. Taubenfeld. Results about fast mutual exclusion. In *Proceedings of the 13th IEEE Real-Time Systems Symposium*, pages 12–21, December 1992.

[BL93] J. Burns and A. Lynch. Bounds on shared memory for mutual exclusion. *Information and Computation*, 107:171–184, 1993.

[DHW93] C. Dwork, M. Herlihy, and O. Waarts. Contention in shared memory algorithms. In *Proc. 25th ACM Symp. on Theory of Computing*, pages 174–183, May 1993.

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

[FMRT90] M. J. Fischer, S. Moran, S. Rudich, and G. Taubenfeld. The wakeup problem. In *Proc. of the 22nd ACM Ann. Symp. on Theory of Computing*, pages 106–116, October 1990. To appear in *SIAM Journal on Computing.*

[FMT93] M. J. Fischer, S. Moran, and G. Taubenfeld. Space-efficient asynchronous consensus without shared memory initialization. *Information Processing Letters*, 45:101–105, February 1993.

[Her91] M. Herlihy. Wait-free synchronization. *ACM Trans. on Programming Languages and Systems*, 13(1):124–149, January 1991.

[Kes82] J.L.W. Kessels. Arbitration without common modifiable variables. *Acta Informatica*, 17:135–141, 1982.

[Lam87] L. Lamport. A fast mutual exclusion algorithm. *ACM Trans. on Computer Systems*, 5(1):1–11, 1987.

[LP90] R. J. Lipton, and A. Park. The processor identity problem. *Information Processing Letters*, 36:91–94, October 1990.

[MS93] M. M. Michael and M. Scott. Fast mutual exclusion, even with contention. Technical Report 460, Department of Computer Science, University of Rochester, June 1993.

[PF77] G. L. Peterson and M. J. Fischer. Economical solutions for the critical section problem in a distributed system. In *Proc. 9th ACM Symp. on Theory of Computing*, pages 91–97, 1977.

[YA93] J-H. Yang and J.H. Anderson. Fast, scalable synchronization with minimal hardware support. In *Proc. 12th ACM Symp. on Principles of Distributed Computing*, 1993. 171–182.

[YA94] J-H. Yang and J.H. Anderson. Time bounds for mutual exclusion and related problems. In *Proc. 26th ACM Symp. on Theory of Computing*, pages 224–233, 1994.